The Adversary Capabilities In Practical Byzantine Fault Tolerance

Yongge Wang
College of Computing and Informatics, UNC Charlotte
Charlotte, NC 28223, USA
yonwang@uncc.edu

September 19, 2021

Abstract

The problem of Byzantine Fault Tolerance (BFT) has received a lot of attention in the last 30 years. 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. In order to address this challenge, researchers have designed randomized BFT protocols in asynchronous networks and deterministic BFT protocols in partial synchronous networks. For both kinds of protocols, a basic assumption is that there is an adversary that controls at most a threshold number of participating nodes and that has a full control of the message delivery order in the network. Due to the popularity of Proof of Stake (PoS) blockchains in recent years, several BFT protocols have been deployed in the large scale of Internet environment. We analyze several popular BFT protocols such as Capser FFG / CBC-FBC for Ethereum 2.0 and GRANDPA for Polkadot. Our analysis shows that the security models for these BFT protocols are slightly different from the models commonly accepted in the academic literature. For example, we show that, if the adversary has a full control of the message delivery order in the underlying network, then none of the BFT protocols for Ethereum blockchain 2.0 and Polkadot blockchain could achieve liveness even in a synchronized network. Though it is not clear whether a practical adversary could *actually* control and re-order the underlying message delivery system (at Internet scale) to mount these attacks, it raises an interesting question on security model gaps between academic BFT protocols and deployed BFT protocols in the Internet scale. With these analysis, this paper proposes a Casper CBC-FBC style binary BFT protocol and shows its security in the traditional academic security model with complete asynchronous networks. Finally, we propose a multi-value BFT protocol XP for complete asynchronous networks and show its security in the traditional academic BFT security model.

Key words: Byzantine Fault Tolerance; security models; blockchain

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 [12] 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 [13] initiated the study of reaching consensus in face of Byzantine failures and designed the first synchronous solution for Byzantine agreement. Dolev and Strong [6] proposed an improved protocol in a synchronous network with $O(n^3)$ communication complexity. By assuming the existence of digital signature schemes and a public-key infrastructure, Katz and Koo [9] proposed an expected constant-round BFT protocol in a synchronous network setting against $\lfloor \frac{n-1}{2} \rfloor$ Byzantine faults.

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 [14] and extended by others such as Cachin, Kursawe, and Shoup [5]. The second category of efforts was to design BFT consensus protocols in partial synchronous networks which was initiated by Dwork, Lynch, and Stockmeyer [7]. Though the network communication model could be different for these protocols, the assumption on the adversary capability is generally same. That is, there is a threshold t such that the adversary could coordinate the activities of the malicious t participating nodes. Furthermore, it is also assumed that the adversary could re-order messages on communication networks.

In recent years, many practical BFT protocols have been designed and deployed at the Internet scale. For example, Ethereum foundation has designed a BFT finality gadget for their Proof of Stake (PoS) blockchain. The current Ethereum 2.0 beacon network uses Casper Friendly Finality Gadget (Casper FFG) [4] and Ethereum foundation has been advocating the "Correct-by-Construction" (CBC) family consensus protocols [19, 20] for their future release of Ethereum blockchain. Similarly, the Polkadot blockchain deployed their home-brew BFT protocol GRANDPA [16]. The analysis in this paper shows that these protocols have an assumption that the adversary cannot control the message delivery order in the underlying networks. Our examples show that if the adversary could control the the message delivery order, then these blockchains could not achieve liveness property. This brings up an interesting question to the research community: what kind of models are appropriate for the Internet scale BFT protocols? Does an adversary have the capability to co-ordinate/control one-third of the participating nodes and to reschedule message delivery order for a blockchain at Internet scale?

Before we have a complete understanding about the impact of the new security assumptions for these blockchain BFT protocols (i.e., the adversary cannot control the message delivery order on the underlying networks), we should still design practical large-scale BFT protocols that are robust in the traditional academic security model. For complete asynchronous networks, we present an Casper CBC-FBC style binary BFT protocol and a multi-value BFT protocol XP and prove their security in the traditional security model.

The structure of the paper is as follows. Section 2 introduces system models and Byzantine agreement. Section 3 shows that Ethereum blockchain 2.0's BFT protocol Casper FFG could not achieve liveness if the adversary can re-order messages in the network. Section 4 shows that Ethereum blockchain's candidate BFT protocol Casper FBC for future deployment could not achieve liveness if the adversary can re-order messages in the network. Section 4 also proposes a Casper FBC style binary BFT protocol that achieves both safety and liveness in the traditional academic security model for complete asynchronous networks. Section 5 reviews the Polkadot's GRANDPA BFT protocol and shows that it cannot achieve liveness if the adversary is allowed to reschedule the message delivery order in the underlying networks. Section 6 proposes a multi-value BFT protocol XP for complete asynchronous networks and proves its security.

2 System model and Byzantine agreement

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 consider three kinds of networks: synchronous networks, partial synchronous networks by Dwork, Lynch, and Stockmeyer [7], and complete asynchronous networks by Fischer, Lynch, and Paterson [8].

1. In a synchronous network, the time is divided into discrete units called slots $T_0, T_1, T_2, \cdots$ where the length of the time slots are equal. Furthermore, we assume that: (1) the current time slot is determined by a publicly-known and monotonically increasing function of current time; and (2) each participant has access to the current time. In a synchronous network, if an honest participant P_1 sends a message m to a participant P_2 at the start of time slot T_i, the message m is guaranteed to arrive at P_2 at the end of time slot T_i.

2. In partial synchronous networks, the time is divided into discrete units as in synchronous networks. The adversary can selectively delay or re-order any messages sent by honest parties. In other words, if an honest participant P_1 sends a message m to an honest participant P_2 at the start of time slot T_{i_1}, P_2 will receive the message m eventually at time T_{i_2} where $i_2 = i_1 + \Delta$. Based on the property of Δ, we can distinguish the following two scenarios:

   - Type I partial synchronous network: $\Delta < \infty$ is unknown. That is, there exists a Δ but participants do not know the exact (or even approximate) value of Δ.
   - Type II partial synchronous network: $\Delta < \infty$ holds eventually. That is, the participant knows the value of Δ. But this Δ only holds after an unknown time slot $T = T_i$. Such a time T is called the Global Stabilization Time (GST).

   For Type I partial synchronous networks, the protocol designer supplies the consensus protocol first, then the adversary chooses her Δ. For Type II partial synchronous networks, the adversary picks the Δ and the protocol designer (knowing Δ) supplies the consensus protocol, then the adversary chooses the GST.

3. In a complete asynchronous network, 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. Thus algorithms based on time-outs cannot be used.

In all of the network models, we assume that the adversary has *complete control of the network*. That is, the adversary may schedule/reorder the delivery of messages as he wishes, and may insert messages as he wishes. 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 Casper the Friendly Finality Gadget (FFG)

Buterin and Griffith [4] proposed the BFT protocol Casper the Friendly Finality Gadget (Casper FFG) as an overlay atop a block proposal mechanism. Casper FFG has been deployed in the Proof of Stake Based Ethereum 2.0. In Casper FFG, weighted participants validate and finalize blocks that are proposed by an existing proof of work chain or other mechanisms. To simplify our discussion, we assume that there are $n = 3t + 1$ validators of equal weight. The Casper FFG works on the checkpoint tree that only contains blocks of height $100 * k$ in the underlying block tree. Each validator P_i can broadcast a signed vote $\langle P_i : s, t\rangle$ where s and t are two checkpoints and s is an ancestor of t on the checkpoint tree. For two checkpoints a and b, we say that $a \to b$ is a supermajority link if there are at least $2t + 1$ votes for the pair. A checkpoint a is justified if there are supermajority links $a_0 \to a_1 \to \cdots \to a$ where a_0 is the root. A checkpoint a is finalized if there are supermajority links $a_0 \to a_1 \to \cdots \to a_i \to a$ where a_0 is the root and a is the direct son of a_i. In Casper FFG, an honest validator P_i should not publish two distinct votes

$$\langle P_i : s_1, t_1\rangle \quad \text{AND} \quad \langle P_i : s_2, t_2\rangle$$

such that either

$$h(t_1) = h(t_2) \quad \text{OR} \quad h(s_1) < h(s_2) < h(t_2) < h(t_1)$$

where $h(\cdot)$ denotes the height of the node on the checkpoint tree. In other words, *an honest validator should neither publish two distinct votes for the same target height nor publish a vote strictly within the span of its other votes*. Otherwise, the validator's deposit will be slashed. The authors [4] claimed that Casper FFG achieves accountable safety and plausible liveness where

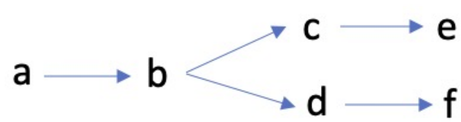
1. achieve accountable safety means that two conflicting checkpoints cannot both be finalized (assuming that there are at most t malicious validators), and
2. plausible liveness means that supermajority links can always be added to produce new finalized checkpoints, provided there exist children extending the finalized chain.

In order to achieve the liveness property, [4] proposed to use the "correct by construction" fork choice rule: the underlying block proposal mechanism should "*follow the chain containing the justified checkpoint of the greatest height*".

The authors in [4] proposed to defeat the long-range revision attacks by a fork choice rule to never revert a finalized block, as well as an expectation that each client will "log on" and gain a complete up-to-date view of the chain at some regular frequency (e.g., once per month). In order to defeat the catastrophic crashes where more than t validators crash-fail at the same time (i.e., they are no longer connected to the network due to a network partition, computer failure, or the validators themselves are malicious), the authors in [4] proposed to slowly drains the deposit of any validator that does not vote for checkpoints, until eventually its deposit sizes decrease low enough that the validators who are voting are a supermajority. Related mechanism to recover from related scenarios such as network partition is considered an open problem in [4].

No specific network model is provided in [4]. In the implementation of the Casper FFG (see GO-Ethereum implementation), a participating node broadcasts his message as soon as he receives a sufficient number of messages to move forward. In other words, even if the network is a synchronized network, a participant may just make his decision on the first $2t+1$ messages and ignore the remaining messages if these first $2t+1$ messages are sufficient for him to move forward. This is reasonable and necessary since the remaining t nodes could be malicious ones and will never send any message at all. Based on this observation, we show that if the adversary could reschedule the message delivery order on the underlying networks, Casper FFG cannot achieve liveness property even in synchronized networks.

Figure 1: Casper FFG cannot achieve liveness



As an example, assume that, at time T, the checkpoint a is finalized where there is a supermajority link from a to its direct child b (that is, b is justified) and no vote for b's descendant checkpoint has been broadcast by any validator yet (see Figure 1). Now assume that the underlying block production mechanism produces a fork starting from b. That is, b has two descendant checkpoints c and d. The adversary who controls the network can arrange t honest validators to receive c first and $t+1$ honest validators to receive d first where $h(c) = h(d)$. Thus t honest validators vote for $b \rightarrow c$, $t+1$ honest validators vote for $b \rightarrow d$, and t malicious validators vote randomly so that both $b \rightarrow c$ and $b \rightarrow d$ receives same number of votes. This means that c and d could not be finalized since neither the link $b \rightarrow c$ nor the link $b \rightarrow d$ could get a supermajority vote. It should be noted that by the two slashing rules in Casper FFG, an honest validator who voted for $b \rightarrow c$ is allowed to vote for $b \rightarrow f$ later since the two votes on $b \rightarrow c$ and $b \rightarrow f$ are not slashable. Next assume that the adversary schedules the message delivery order so that t honest validators receive e first and $t+1$ honest validators receive f first (without loss of generality, we may assume that $h(e) = h(f)$). Thus t honest validators vote for $b \rightarrow e$, $t+1$ honest validators vote for $b \rightarrow f$, and t malicious validators vote randomly so that both $b \rightarrow e$ and $b \rightarrow f$ receives same number of votes. Thus e and f could not be finalized since neither the link $b \rightarrow e$ nor the link $b \rightarrow f$ could get a supermajority vote. This process continues forever and no checkpoint after a could be finalized. That is, Casper FFG could not achieve liveness with this kind of message delivery schedule by the adversary.

4 CBC Casper the Friendly Binary Consensus (FBC)

The network model for Casper FFG is not clearly defined. In order to make Ethereum blockchain robust in complete asynchronous networks, Ethereum foundation has been advocating the "Correct-by-Construction" (CBC) family of Casper blockchain consensus protocols [19, 20] for their future release of Ethereum 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 [19] "*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., [15]). 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 [20] 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 section 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.

4.1 Casper FBC protocol description

CBC Casper contains a binary version and an 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:

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

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

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

4.2 Efforts to achieve liveness for 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 [15]) 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.

4.3 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 [19, 20], 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 [19, 20] since there is zero equivocating participant.

4.4 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:

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

As we have mentioned in the preceding section, CBC Casper FBC [19, 20] 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:

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

4.5 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 can be transformed to fail-stop participants. In the following, we assume that a reliable broadcast primitive such as the one by Bracha is used in our protocol execution and present Bracha's style BFT protocol in 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 [14] 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 a common coin for the above BFT protocol. Thus the above BFT protocol could be implemented with constant steps on Ethereum 2.0.

5 Polkadot's BFT protocol GRANDPA

The project Polkadot (`https://github.com/w3f`) proposed an algebraic approach based BFT finality gadget protocol GRANDPA which is similar to Casper FBC in some sense. There are different versions of GRANDPA protocol. In this paper, we refer to the most recent one [16] dated on June 19, 2020. Specifically, Polkadot implements a nominated proof-of-stake (NPoS) system. At certain time period, the system elects a group of validators to serve for block production and the finality gadget. Nominators also stake their tokens as a guarantee of good behavior, and this stake gets slashed whenever their nominated validators deviate from their protocol. On the other hand, nominators also get paid when their nominated validators play by the rules. Elected validators get equal voting power in the consensus protocol. Polkadot uses BABE as its block production mechanism and GRANDPA as its BFT finality gadget. Here we are interested in the finality gadget GRANDPA (GHOST-based Recursive ANcestor Deriving Prefix Agreement) that is implemented for the Polkadot relay chain. GRANDPA contains two protocols, the first protocol works in partially synchronous networks and tolerates 1/3 Byzantine participants. The second protocol works in full asynchronous networks (requiring a common random coin) and tolerates 1/5 Byzantine participants. The first GRANDPA protocol assumes that underlying network is a Type I partial synchronous network. In the following paragraphs, we will show that GRANDPA cannot achieve liveness property in partial synchronous networks if the adversary is allowed to reschedule the message delivery order.

Assume that there are $n = 3t + 1$ participants $P_0, \cdots, P_{n-1}$ and at most t of them are malicious. Each participant stores a tree of blocks produced by the block production mechanism with the genesis block as the root. A participant can vote for a block on the tree by digitally signing it. For a set S of votes, a participant P_i equivocates in S if P_i has more than one vote in S. A set S of votes is called safe if the number of participants who equivocate in S is at most t. A vote set S has supermajority for a block B if

$$|\{P_i : P_i \text{ votes for } B*\} \cup \{P_i : P_i \text{ equivocates}\}| \geq 2t + 1$$

where P_i votes for $B*$ mean that P_i votes for B or a descendant of B.

In GRANDPA, the 2/3-GHOST function $g(S)$ returns the block B of the maximal height such that S has a supermajority for B or a "nil" if no such block exists. If a safe vote set S has a supermajority for a block B, then there are at least $t + 1$ voters who do vote for B or its descendant but do not equivocate. Based on this observation, it is easy to check that if $S \subseteq T$ and T is safe, then $g(S)$ is an ancestor of $g(T)$.

The authors in [16] defined the following concept of *possibility* for a vote set to have a supermajority for a block: "We say that it is *impossible* for a set S to have a supermajority for a block B if at least $2t + 1$ voters either equivocate or vote for blocks who are not descendant of B. Otherwise it is *possible* for S to have a supermajority for B". Then they claimed (the second paragraph above Lemma 2.6 in [16]) that "a vote set S is possible to have a supermajority for a block B if and only if there exists a safe vote set $T \supseteq S$ such that T has a supermajority for B". Unfortunately, this claim is not true in practice if the adversary selects a non-equivocating strategy which may introduce deadlock to the system (on the other hand, this claim is true if all t malicious voters MUST equivocate).

Example 5.1 *Assume that blocks C and D are inconsistent (that is, C is not an ancestor of D and D is not an ancestor of C) and the vote set S contains the following votes which could be achieved by letting the adversary re-order the messages to be delivered on the network (this could also happen before GST in partial synchronous networks).*

1. $t + 1$ *voters vote for* C.
2. $2t$ *voters vote for* D.
3. *no voters equivocate.*

Since only $2t$ votes in S that "either equivocate or vote for blocks who are not descendant of C", by the above definition, S is NOT impossible *to have a supermajority for C. Thus, by the above definition, S is* possible *to have a supermajority for a block C. If malicious voters choose not to equivocate (we cannot force a malicious voter to equivocate), there does not exist a semantically valid safe vote set $T \supseteq S$ such that T has a supermajority for C. Similarly, by the above definition, S is NOT* impossible *to have a supermajority for D and is* possible *to have a supermajority for a block D. If malicious voters choose not to equivocate, there does not exist a semantically valid safe vote set $T \supseteq S$ such that T has a supermajority for D. On the other hand, if a malicious voter submits another vote (either for C or D) to S, then D has a supermajority vote in S.*

In the following sections, we will use Example 5.1 to show that the GRANDPA protocol will enter deadlock and cannot achieve the liveness property if the adversary is allowed to reschedule the message delivery order.

5.1 GRANDPA protocol

The GRANDPA protocol starts from round 1. For each round, one participant is designated as the primary and all participants know who is the primary. Each round consists of two phases: *prevote* and *precommit*. Let $V_{r,i}$ and $C_{r,i}$ be the sets of prevotes and precommits received by P_i during round r respectively. Let $E_{0,i}$ be the genesis block and $E_{r,i}$ be the last ancestor block of $g(V_{r,i})$ that is possible for $C_{r,i}$ to have a supermajority. If either $E_{r,i} < g(V_{r,i})$ or it is impossible for $C_{r,i}$ to have a supermajority for any children of $g(V_{r,i})$, then we say that P_i sees that round r is *completable*. Let Δ be a time bound such that it suffices to send messages and gossip them to everyone. The protocol proceeds as follows.

1. P_i starts round $r > 1$ if round $r - 1$ is completable and P_i has cast votes in all previous rounds. Let $t_{r,i}$ be the time P_i starts round r.
2. The primary voter P_i of round r broadcasts $E_{r-1,i}$.
3. **prevote**: P_i waits until either it is at least time $t_{r,i} + 2\Delta$ or round r is completable. P_i *prevotes* for the head of the best chain containing $E_{r-1,i}$ unless P_i receives a block B from the primary with $g(V_{r-1}, i) \geq B > E_{r-1,i}$. In this case, P_i uses the best chain containing B.
4. **precommit**: P_i waits until $g(V_{r,i}) \geq E_{r-1,i}$ and one of the following holds
   (a) it is at least time $t_{r,i} + 4\Delta$
   (b) round r is completable

   Then P_i broadcasts a precommit for $g(V_{r,i})$

At any time after the precommit step of round r, if P_i sees that $B = g(C_{r,i})$ is descendant of the last finalized block and $V_{r,i}$ has a supermajority, then P_i finalizes B.

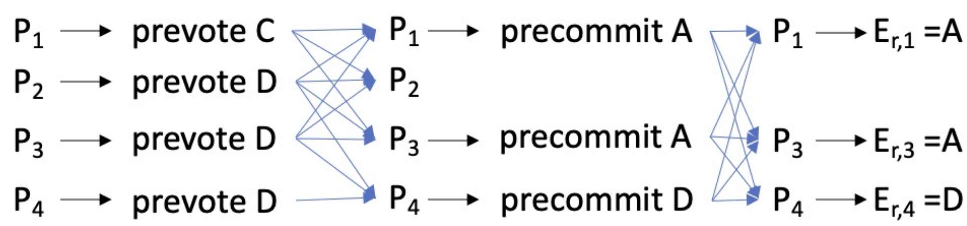
5.2 GRANDPA cannot achieve liveness in partial synchronous networks

In this section, we show that GRANDPA BFT protocol cannot achieve liveness property in partial synchronous networks. Assume that $E_{r-1,0} = \cdots = E_{r-1,n-1} = A$ and all $3t + 1$ voters prevote and precommit to A during round $r - 1$ and A is finalized by all voters during round $r - 1$. Also assume that no voter will ever equivocate. During round r, the block production mechanisms produces a fork of of A. That is, we get two children blocks C and D of A.
Counter-example 1: By adjusting the message delivery schedule (this could happen before GST in partial synchronous networks), $t + 1$ voters only receive the block C before time $t_{r,i} + 2\Delta$ and $2t$ voters only receive the block D before time $t_{r,i} + 2\Delta$. However, all voters will receive both blocks C and D before time $t_{r,i} + 3\Delta$.

At step 2 of round r, the primary voter broadcasts $A = E_{r-1,i}$. At step 3, both $V_{r,i}$ and $C_{r,i}$ are empty initially, the round r cannot be completable until time $t_{r,i} + 2\Delta$. Thus voter P_i waits until time $t_{r,i} + 2\Delta$ to submit its prevote. The $t + 1$ voters that received block C would prevote for C and the other $2t$ voters that received block D would prevote for D. The adversary allows all prevotes of Step 3 to be delivered to all voters synchronously before time $t_{r,i} + 4\Delta$. During Step 4, each voter P_i receives $t + 1$ prevotes for C and $2t$ prevotes for D. Since $C_{r,i}$ is empty until it receives any precommit, round r is not completable until time $t_{r,i} + 4\Delta$. That is, each voter P_i waits until $t_{r,i} + 4\Delta$ to precommit $g(V_{r,i}) = A$. The adversary allows all voters to receive all precommit votes for A. Now each voter P_i estimates $E_{r,i} = g(V_{r,i}) = A$. By the fact that $C_{r,i} = \{3t + 1 \text{ precommit votes for } A\}$, we have $g(C_{r,i}) = A$. Since A has already been finalized, P_i will not finalize any block during round r.

In order for the round r to be completable, we need "either $E_{r,i} < g(V_{r,i})$ or it is impossible for $C_{r,i}$ to have a supermajority for any children of $g(V_{r,i})$". However, we have $E_{r,i} = g(V_{r,i}) = A$ and $C_{r,i} = \{3t + 1 \text{ precommit votes for } A\}$. That is, by definition of "*possibility*", it is "**possible**" for $C_{r,i}$ to have a supermajority for both children C and D of $g(V_{r,i}) = A$. In order words, the round r is NOT "completable" and GRANDPA cannot start Step 1 of round $r + 1$.

Figure 2: Counter-example 2 for GRANDPA



Counter-example 2: This example is more involved than counter-example 1 and an example with $t = 1$ is shown in Figure 2. By adjusting the message delivery schedule (this could happen before GST in partial synchronous networks), by time $t_{r,i} + 2\Delta$, we have t voters received block C and $2t + 1$ voters received block D. Furthermore, all voters will receive both blocks C and D before time $t_{r,i} + 3\Delta$.

At step 2 of round r, the primary voter broadcasts $A = E_{r-1,i}$. At step 3, both $V_{r,i}$ and $C_{r,i}$ are empty initially, the round r cannot be completable until time $t_{r,i} + 2\Delta$. Thus voter P_i waits until time $t_{r,i} + 2\Delta$ to submit its prevote. The t voters that received block C would prevote for C and the other $2t + 1$ voters that received block D would prevote for D. Durng Step 4, the adversary schedules the message delivery in such a way that, by time $t_{r,i} + 4\Delta$, t voters receive "$2t + 1$ prevotes for D" and $2t + 1$ voters receive "t prevotes for C and $t + 1$ prevotes for D". Since $C_{r,i}$ is empty until it receives any precommit, round r is not completable until time $t_{r,i} + 4\Delta$. That is, each voter P_i waits until $t_{r,i} + 4\Delta$ to precommit $g(V_{r,i})$. At time $t_{r,i} + 4\Delta$, t voters precommit for $D = g(V_{r,i})$, $2t$ voters precommit for $A = g(V_{r,i})$, and one malicious voter does not precommit. The adversary let all precommit messages to be delivered to all voters synchronously.

Now t voters estimates $E_{r,i} = g(V_{r,i}) = D$ and $2t + 1$ voters P_i estimates $E_{r,i} = g(V_{r,i}) = A$. By the fact that

$$C_{r,i} = \{t \text{ precommit votes for } D \text{ and } 2t \text{ precommit votes for } A\},$$

we have $g(C_{r,i}) = A$. Since A has already been finalized, P_i will not finalize any block during round r. In order for the round r to be completable, we need "either $E_{r,i} < g(V_{r,i})$ or it is impossible for $C_{r,i}$ to have a supermajority for any children of $g(V_{r,i})$". However, we have $E_{r,i} = g(V_{r,i})$ for all voters and, by Example 5.1, it is "**possible**" for $C_{r,i}$ to have a supermajority for all children of $g(V_{r,i})$. In order words, the round r is NOT "completable" and GRANDPA cannot start Step 1 of round $r + 1$.

Paper [16, page 7] mentions that "*$C_{r,i}$ and $V_{r,i}$ may change with time and also that $E_{r-1,i}$, which is a function of $V_{r-1,i}$ and $C_{r-1,i}$, can also change with time if P_i sees more votes from the previous round*". However, this has no impact on our preceding examples since after an honest voter prevotes/precommits, the honest voter cannot change his prevote/prevommit votes anymore (otherwise, it will be counted as equivocation).

6 Multi-value BFT protocols for asynchronous networks

Section 4.5 proposed a binary BFT finality gadget in complete asynchronous networks and Section **??** proposed a multi-value BFT finality gadget for partial synchronous networks. Furthermore, the BFT protocol in Section 4.5 requires a strongly reliable broadcast channel. In this section, we present a constant round multi-value BFT protocol XP for complete asynchronous networks that does not require strongly reliable broadcast channels. The XP protocol is motivated by the probabilistic binary BFT protocol in Cachin, Kursawe, and Shoup [5] and requires a shared common random beacon which could be implemented using the Ethereum random beacon.

Similar to Section **??**, we assume that there is a partial order on the list of candidate blocks to be finalized: $\mathcal{B} = \{B_j : 1 \leq j \leq \tau\}$ where $B_1 \prec B_2 \prec \cdots \prec B_\tau$. During the protocol run, each participant P_i maintains a list of known candidate blocks in its local variable $X_i \subseteq \mathcal{B}$. At the start of the protocol run, X_i contains the list of candidate blocks that the participant P_i has learned and could be empty. During the protocol run, we assume that there is a random coin shared by all participants. For example, for the Ethereum 2.0, one may use the existing random beacon protocol as a common coin. Let σ be the random string shared by all participants for step s. Then participant P_i sets the "common" block X_i^σ as a block $B_j \in X_i$ such that $H(B_j, s)$ and $H(\sigma, s)$ has the maximal common prefix within X_i, where $H(\cdot)$ is a hash function. If there are two candidate blocks $B_{j_1} \prec B_{j_2}$ such that

$$\text{commonPrefix}(H(B_{j_1}, s), H(\sigma, s)) = \text{commonPrefix}(H(B_{j_2}, s), H(\sigma, s)),$$

then P_i sets $X_i^\sigma = B_{j_2}$. It is easy to observe that if $X_{i_1} = X_{i_2}$, then $X_{i_1}^\sigma = X_{i_2}^\sigma$. However, if $X_{i_1} \neq X_{i_2}$, then $X_{i_1}^\sigma$ and $X_{i_2}^\sigma$ may be different.

The protocol proceeds from step to step until an agreement is achieved and the protocol does not have any assumption on the time setting. Each participant waits for at least $n - t$ justified messages from participants (including himself) to proceed to the next sub-step. The step $s \geq 0$ for a participant P_i consists of the following sub-steps:

- **lock**: If $s = 0$, then let B be the maximal element in X_i. If $s > 0$ then wait for $n - t$ justified commit-votes from step $s - 1$ and let

$$B = \begin{cases} B' & P_i \text{ receives a commit-vote for } B' \text{ in step } s-1 \\ X_i^\sigma & P_i \text{ receives } 2t+1 \text{ commit-votes for } \perp \text{ and } \sigma \text{ is common coin} \end{cases} \tag{6}$$

  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 messages to justify the selection of the value B.

- **commit**: P_i collects $n - t$ justified `lock` messages (7) and 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 messages to justify the selection of the value $\bar{B}$.

- **check-for-decision**: Collect $n - t$ properly justified commit votes (9) and lets $X_i = X_i \cup (\cup_j X_j)$ where X_j are from messages (9). 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 by establishing a sequence of lemmas.

Lemma 6.1 *If an honest participant P_i decides on the value $\bar{B}$ at the end of step s (but no honest participant has ever decided before 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 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. This means that, all honest participants will commit-vote for the block $\bar{B}$ during step $s + 1$ and any commit-vote for other blocks cannot be justified. Thus, all honest participants will collect $n - t$ justified commit-vote for the block $\bar{B}$ and decide on block $\bar{B}$ at the end of step $s + 1$. □

Lemma 6.2 *Block B in equation (6) is uniquely defined for each honest participant.*

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 $2t + 1$ participants who submit `lock` messages for $\bar{B}_1$ and $2t + 1$ participants who submit `lock` messages for $\bar{B}_2$. This means that at least $t + 1$ participants (thus at least one honest participant) submit `lock` messages for both $\bar{B}_1$ and $\bar{B}_2$ which is impossible. □

Lemma 6.3 *During step s, if participants P_i and P_j receive commit votes for $\bar{B}_1$ and $\bar{B}_2$ respectively, then $\bar{B}_1 = \bar{B}_2$.*

Proof. For a contradiction, assume that $\bar{B}_1 \neq \bar{B}_2$. Then there are $2t+1$ lock messages for $\bar{B}_1$ and $2t+1$ lock messages for $\bar{B}_2$ during step s. This means that at least $t+1$ participants (thus at least one honest participant) submit `lock` messages for both $\bar{B}_1$ and $\bar{B}_2$ which is impossible. □

Lemma 6.4 *If all honest participants hold the the same local value $X_i = \mathcal{B}$ at the start of step s, then with high probability, every participant decides by the end of step $s + \tau$.*

Proof. The Lemma is proved by distinguishing the following two cases:

1. $s = 0$: At step 0, each honest participant broadcasts the lock for B_τ though dishonest participant may broadcast a lock for another block. At the commit phase, each honest participant P_i broadcasts $\mathcal{B}$ and a commit message for $\perp$ or B_τ depending on what he receives. If some participant decides at the end of Step 0, by Lemma 6.1, all honest participants decide by the end of Step 1. Assume that no participant decides by the end of Step 0. During Step 1, when a participant broadcasts a lock for a block, he needs to include $2t + 1$ commit messages from Step 0 as the justification. Among these $2t + 1$ commit messages, at least $t + 1$ come from honest participants which contain $\mathcal{B}$. Thus from now on, each participant must include its local variable $X_i = \mathcal{B}$ in its justification message. In other words, if a participant broadcast a lock for a block based on the common coin, this locked block must be identical for all participants who use the common coin. Therefore, from Step 1 on, a participant can only broadcast a lock for a block committed in the immediate previous step (cf. Lemma 6.3) or a lock for a block determined by the common coin. With probability $\frac{1}{\tau}$, the block determined by the common coin is identical to the committed block from the previous step. Thus all honest participants are expected to decide by Step τ.

2. $s > 0$: for this case, we distinguish the following three cases:

   (a) By the end of step $s - 1$, at least one participant (including dishonest participant) can legally decide on a block (this means at least one honest participant receives a commit-vote for a block $B \neq \perp$ during step $s - 1$): By Lemma 6.1, all honest participants decides by the end of step s.

   (b) By the end of step $s - 1$, no participant (including dishonest participant) can legally decide on a block: From Step s and on, each honest participant broadcasts a lock message for the unique block X_i^σ determined by the common coin or a unique block that was committed in the immediate previous Step (cf. Lemma 6.3). With probability $\frac{1}{\tau}$, the block determined by the common coin is identical to the committed block from the immediate previous Step. Thus all honest participants are expected to decide by Step $s + \tau$.

This completes the proof of the Lemma. □

Lemma 6.5 *All honest participant decides in constant steps.*

Proof. If no participant decides by the end of Step $s+\tau$, then, by Lemma 6.4, with high probability, at least one honest participant P_i revises its local variable X_i to include at least one more element during the Steps from s to $s+\tau$. Since there are at most τ candidate blocks, this process continues until no honest participant revises its local variable X_i. Then, by Lemma 6.4, all honest participants hold the same candidate block and the consensus will be reached. □

The above five 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.

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] D. Dolev and H.R. Strong. Polynomial algorithms for multiple processor agreement. In *Proc. 14th ACM STOC*, pages 401–407. ACM, 1982.

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

[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] J. Katz and C.-Y. Koo. On expected constant-round protocols for byzantine agreement. *Journal of Computer and System Sciences*, 75(2):91–112, 2009.

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

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

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

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

[16] A. Stewart and E. Kokoris-Kogia. GRANDPA: a byzantine finality gadge `https://github.com/w3f/consensus/blob/master/pdf/grandpa.pdf`, June 19, 2020.

[17] Wang, Yongge. Byzantine Fault Tolerance in Partially Connected Asynchronous Networks. IACR Cryptol. ePrint Arch., 2019. `https://eprint.iacr.org/2019/1460.pdf`

[18] Wang, Yongge. Blockchain BFT Protocol for Complete Asynchronous Networks. 2020. arXiv preprint arXiv:2005.04309. `https://arxiv.org/pdf/2005.04309.pdf`

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

[20] 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 strongly reliable 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.