Lecture 15: Permissionless Blockchains based on BFT Protocols: Algorand

Principles of Blockchains, University of Illinois,
Professor: Pramod Viswanath
Scribe: Gerui Wang

March 18, 2021

Abstract

In the previous lecture, we saw BFT protocols that provide consensus with *finality*, i.e., deterministic safety guarantees. However, the protocols we saw (e.g., Hotstuff) were for a fixed number of participants. In this lecture, we study how to use a BFT protocol (such as Hotstuff) in the context of a permissionless blockchain. We pay particular attention to participation determined by PoS, and security against adaptive adversaries. We study Algorand, a BFT protocol that is very efficient in throughput and latency and is especially suited to operation in a permissionless setting via Proof of Stake (PoS), providing resistance against an adaptive adversary.

From Permissioned to Permissionless: Committee Selection

Consider a very large number n of nodes, but with variable (unknown) participation level, i.e., not all the nodes may be online. This is a bit more constrained than the truly permissionless setting, where n itself is unknown. A simple way to convert a permissioned BFT protocol to a permissionless one is to simply select a committee of fixed size (N) and run a BFT protocol (e.g., HotStuff) to reach consensus on a new block. The randomness of committee election comes from the previous block. Now, as long as 2/3 of this random committee is *online and honest*, the BFT protocol is safe and live. Due to the randomness in the committee election, we need to allow some slack in the condition on the fraction of honest players. For example, we can achieve 2/3 honest supermajority of committee when the honest fraction is 0.7. Given the committee size N, the random variable of honest committee member X follows Binomial distribution Binomial$(N, 0.7)$. And figure 1 shows the probability $P(X > 2N/3)$, and we can see as N grows larger, the probability approaches 1.

However, it is difficult to elect a fixed committee size in a distributed verifiable manner. But it is easy to do a random committee size, using hash functions (like we did in Lecture 12). Each player (honest or malicious) runs a lottery with a small probability of being elected. So now N is also random, but with a fixed expectation. For instance, there are 700K honest players and 300K malicious (Byzantine) players in total, and let $E[N] = 1000$. Then the lottery's probability of being elected is 0.001 for every player. And X follows Binomial(700K,0.001) and Y follows Binomial(300K,0.001) (Y is the random variable of malicious committee member).

For a committee with a random size, besides requiring $X > 2E[N]/3$, which ensures that honest players can make progress (liveness), we also need $X + 2Y < 4E[N]/3$ which ensures the committee has 2/3 honest supermajority (safety). Now, we can re-plot the probability in figure 2 and we see again that the probability of a random committee leading to a secure BFT protocol approaches 1 rapidly.

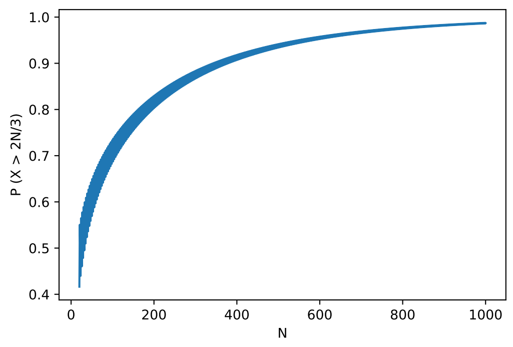


Figure 1: Probability of honest supermajority for fixed committee size N.

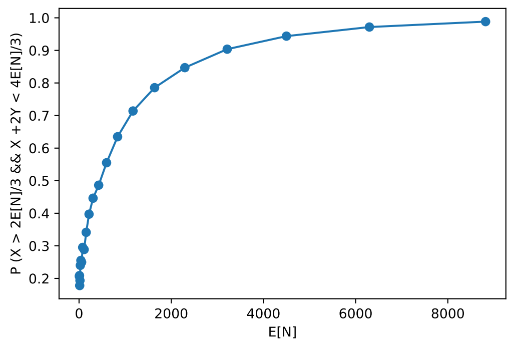


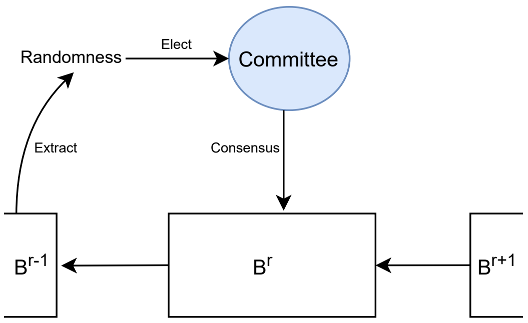
Figure 2: Probability of liveness and honest supermajority for a random committee size N. Total players $n = 1,000,000$ and honest fraction 0.7.

This form of random committee selection is illustrated in figure 3a. However, insecurity arises due to vulnerability to adaptive adversaries. The vulnerability is of two kinds: (a) the committee selection is known ahead of time and thus an adaptive adversary can corrupt the committee beforehand; (b) even within a committee, individual nodes have specific roles in the context of the BFT protocol and can get corrupted during the BFT operation. Algorand addresses both these problems by: (a) novel committee selection that is robust to an adaptive adversary (using secret credentials); (b) novel BFT consensus that is robust to adaptive adversary even within substeps of the protocol, as depicted in figure 3b. How these are implemented is the topic of the next two sections.

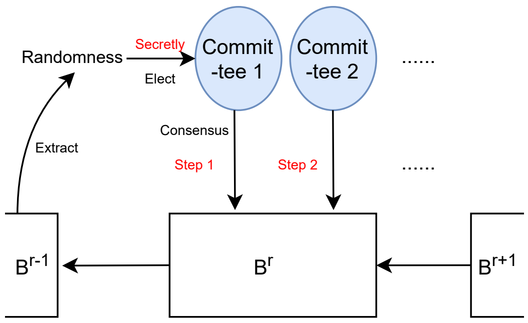
Player Replaceability and Secret Committee Election

The existence of an adaptive adversary requires each substep of a BFT protocol to be assigned to a totally new committee, which is independently and randomly selected among all players. This property is called *Player Replaceability*. In addition, since an adversary can corrupt the committee as soon as it knows their identities, we require the committee election to be held *secretly*. In this section, we show how Algorand achieves these requirements.

Suppose the block B^{r-1} associated with round $r-1$'s is available on the blockchain, and we are running a BFT protocol to determine block B^r. Assume that we can obtain a random quantity Q^r from the last block B^{r-1}, which will be used to generate the randomness. The BFT protocol for



(a) Simple committee



(b) Algorand

Figure 3: Simple committee selection vs Algorand committee selection

block B^r involves multiple steps, and we denote the step counter by s. Algorand elects different committees for each step, as seen below.

In any committee selection stage, player i uses a quantity known as *credential* (σ) to secretly determine whether it has been selected. Relying on the random quantity Q^r that is deduced from block B^{r-1} (assuming round $r-1$'s block B^{r-1} available on the blockchain), the credential $\sigma_i^{r,s}$ is a unique signature on r, s, Q^r and if $H(\sigma_i^{r,s}) < p$ (H is a hash function, p is a threshold) then player i is selected as a committee member. The unique secret signature is generated using VRFs we have introduced in Lecture 12: $VRF(r, s, Q^r, sk_i) < p$ where sk_i is the secret key for VRF. The threshold p is chosen to have a suitable expected committee size.

When its time to act in step s arrives, i propagates $\sigma_i^{r,s}$ with its message so that other players can verify its inclusion in the committee. Although the adversary may corrupt it, the message cannot be stopped from reaching other honest players. Moreover, the adversary has no more control on the rest of the protocol than he has by corrupting a random player: the committees of all future steps will be randomly and independently selected.

Ephemeral keys. Although the adversary cannot predict beforehand which users will be in the committee, it would know their identities after seeing their messages, and could then corrupt all of them and oblige them to certify a fake block. To prevent this, players use ephemeral keys: public/secret key pairs that are single-use-only, and once used, are destroyed. Only credentials σ are signed by the long-term public/secret key pairs, and any other message is signed by ephemeral key pairs.

To generate an ephemeral key pair for a player-round-step triple (i, r, s) player i first generates a master key pair, then it uses the master key pair to generate the ephemeral key pairs for multiple rounds and steps, after which it destroys the master secret key and publicizes the master public key. At round r, step s, if player i is elected as a committee member, it uses the ephemeral key for (i, r, s) to sign messages, and then destroys the ephemeral secret key. Ephemeral keys are different from the key evolving scheme (KES) we saw in Lecture 12 in that the ephemeral keys are generated by a master key pair, rather than evolving from the previous ephemeral key pair. Since it's very unlikely to be elected as a committee member, a player does not necessarily need to keep keys for every round and step as it does in KES. In this situation, using ephemeral keys is more suitable and more efficient than using KES.

Algorand: Single Round BFT Consensus Protocol

We start with a probabilistic binary BFT consensus protocol that doesn't have player replaceability (but naturally allows its addition, as we see later). This protocol targets $n = 3t + 1$ players and t-Byzantine fault tolerance. Each player i holds a binary value b_i as input, on which they want to reach agreement. During the protocol, they keep updating their local binary value b_i. At the end of the protocol, honest players should output the same value b, that is, there is a value b such that $b_i = b$ for all honest player i. If all honest players hold the same input, that is, there is a value b such that initially $b_i = b$ for all honest players, then at the end they should output b, that is, $b_i = b$ still holds for all honest players.

The protocol proceeds in synchronous steps, where messages are guaranteed to be delivered within a step. So the protocol is designed for a *synchronous* network, like the longest chain protocol. Each step follows the paradigm below:

Start of a step	End of a step
Every player propagates b_i	Every player updates b_i based on the received messages

Intuition

The key idea in the Algorand protocol lies in the way *player i updates b_i*. Denote $\#_i(v)$ to be the number of players from which i has received the value v, possibly including i itself. The update rule is the following:

> If $\#_i(0) \geq 2t + 1$, then player i sets $b_i = 0$. Symmetrically, if $\#_i(1) \geq 2t + 1$, then i sets $b_i = 1$.

The above update rule stipulates that if more than $2t + 1$ players hold 0 (or 1), of which $t + 1$ must be honest, then player i sets b_i to 0 (or 1). Notice that the two conditions $\#_i(0) \geq 2t + 1$ and $\#_i(1) \geq 2t+1$ never happens together in one step, since only Byzantine players vote twice and there are at most t of them.

An obvious issue with the above step is what if $\#_i(0) < 2t + 1$ and $\#_i(1) < 2t + 1$? This condition implies there has not been a 2/3 majority of a value yet. To deal with this situation, we use a cryptographic primitive called *common coin*. Assume in each step there is a new randomly and independently selected bit c shared with all players (we will show how to implement it later). The usage of the common coin is straightforward: each player sets $b_i = c$ when enough of the players have not had consensus. Hence, the updating of b_i becomes:

1. If $\#_i(0) \geq 2t + 1$, then i sets $b_i = 0$.
2. Else, if $\#_i(1) \geq 2t + 1$, then i sets $b_i = 1$.
3. Else, i sets $b_i = c$.

The intuition is to run this step sufficiently many times, say $m = 100$ times, and let player i output its local value b_i. The following analysis shows that with sufficient iterations of this step, consensus is achieved among honest players with finality.

Analysis The core step has the following properties,

(A) If, at the start of a step, the honest players are in agreement on a bit b, (i.e., if $b_i = b$ for all honest player i), then they remain in agreement on b by its end.

(B) If the honest players are not in agreement (on any bit) at the start of a step, then with probability 1/2, they will be in agreement (on some bit) by its end.

Property (A) is easy to see since there are at least $2t+1$ honest players and they propagate their agreed values. The explanation for property (B) is that when honest players are not in agreement, they can be in either condition 1 and 3 (sets $b_i = 0$ and $b_i = c$) or condition 2 and 3 (sets $b_i = 1$ and $b_i = c$). In either case, coin c is equal to the bit with probability $1/2$. Thus, despite initial values, by running this step m times, honest players will reach an agreement with probability $1 - (1/2)^m$. However, this is not a consensus protocol, since honest players are not aware whether they are in agreement or not (even after m steps) and when to terminate. The next section constructs a BFT consensus protocol, utilizing this basic step, to solve this problem.

Consensus on a binary value

Algorand's binary BFT consensus protocol is an ever-running loop that runs three steps in turn. The protocol is shown in figure 4. We also use a counter s (starting with $s = 1$) to represent how many steps it has executed. So the first step has counter $s = 1, 4, \ldots$; the second step $s = 2, 5, \ldots$; and the third step $s = 3, 6, \ldots$.

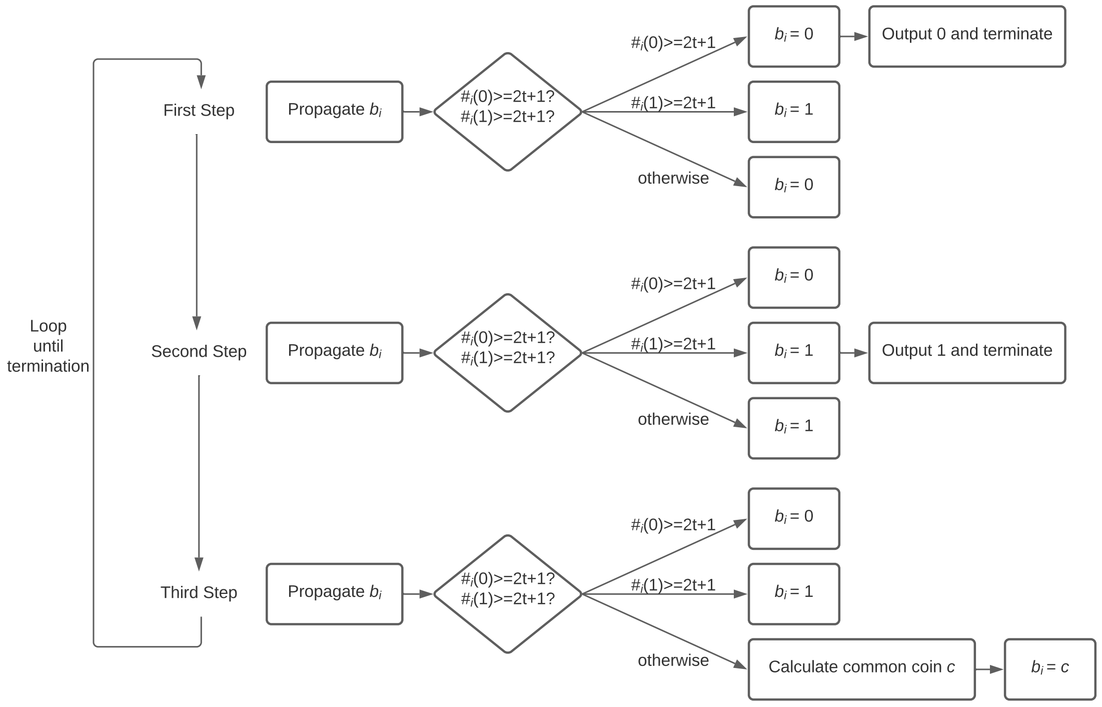


Figure 4: Algorand consensus on a binary value

The first and second steps in the figure have the aforementioned property (A). They also ensure that, once the agreement has already been reached on some bit, an honest player can learn this is the case, output the bit, and terminate. This is formally described as property (C).

(C) If, at the first or second step, an honest player i outputs, then agreement will hold at the end of the step.

As already mentioned, the third step has property (A) and (B). From these properties, we can show that this is a Byzantine consensus protocol. (A) and (C) ensure that no honest player outputs different values, and (A) and (B) ensures that they eventually reach agreement and output a value.

Adding player replaceability and secret committee election

To add the committee into the protocol, we give the eligibility to propagate messages to the committee. Notice that updating local value b_i is still done by all players. The communication complexity is $O(nN)$ since each committee member sends messages to all players.

To enable player replaceability and secret committee election, players calculate credentials as introduced in the previous section. Once player i calculates its credential for round r, step s: $\sigma_i^{r,s}$, it learns its role (whether in committee), and can prove its role by appending the credential to its messages in this step. In short, within a step:

Start of a step	End of a step
Committee member propagates b_i appends credential $\sigma_i^{r,s}$	**Every player** updates b_i based on the received messages

In the protocol illustrated in Figure 4, we simply need to replace the quantity $2t+1$ by $2E[N]/3$, since now the random committee has expected size $E[N]$ rather than $n = 3t+1$.

Implementing common coin

Here we show how to implement the common coin in the third step. The common coin is generated in a separate step where a player receives messages from many players, denoted by a set SV. The player picks the smallest credential hash from SV, hashes the credential with the step counter s, and uses the least significant bit as the coin c. Formally,

> Letting SV be the set of players from whom it has received a valid message, and letting $l = \arg\min_{j \in SV} H(\sigma_j)$. i uses $\texttt{lsb}(H(\sigma_j, s))$ as the common coin, where H is a hash function and `lsb` is the least significant bit.

Analysis. If player l whose credential hash is smallest is honest, then this procedure implements the common coin. Player l is honest with probability $2/3$. Hence, this procedure generates a correct common coin for all players with probability $2/3$. Therefore, the property (B) becomes "with probability $1/3$ (rather than $1/2$), honest players will be in agreement". The protocol terminates within constant steps in expectation.

Multivalue consensus

Here we present the intuition on extending binary value to multivalue Algorand consensus. First, we need to elect a leader for each round r who should build and propagate a valid block B^r. The "potential leader" election is the same as in the committee election, except with a much smaller threshold to elect just a few dozen players as potential leaders. Then (possibly multiple) potential leaders propagate their blocks. Each player chooses the leader whose credential hash is the smallest from potential leaders, and vote for the block hash twice: only when receiving more than $2/3$ first votes should it send the second vote. After these two steps (block proposing and two-round voting), each player either (a) receives a valid block B^r from the leader and enough votes for its hash, or (b) no valid block has enough votes for its hash.

Then, player i starts the binary protocol with initial value $b_i = 0$ in condition (a) or $b_i = 1$ in condition (b). In the binary protocol, players also attach the block hash to their messages as additional information in condition (a), to make sure they hold the same block. The two-round voting also ensures that if two honest players start with condition (a), then they receive the same block B^r. If the binary protocol outputs value 0, it means block B^r (which is known to all players) is finalized. Otherwise, if the output is value 1, it means an empty block B_ϵ^r is finalized.

Notice that the aforementioned block proposing and two-round voting steps are also committee-based and player-replaceable. Therefore, the whole Algorand protocol is committee-based and player-replaceable.

Conclusion

Putting the steps together, we have Algorand consensus where players agree on block B^r for round r. To build the Algorand Blockchain, the consensus protocol is executed round by round to generate block $B^1, B^2, \ldots$ sequentially, where block B^{r+1} contains a hash pointer to parent block B^r. Each round has $O(nN)$ communication and terminates in constant expected steps, as shown in previous sections.

We can also turn Algorand into a proof-of-stake (PoS) blockchain, if we involve the stake held by a public key into the committee/leader election (more stake, higher chance to be elected).Since there is no forking at all in Algorand, none of the key grinding and nothing-at-stake attacks that we saw in the longest chain version of PoS (Lecture 12) arise here. So the individual block security of Algorand guarantees security of the whole blockchain. One weakness is that the security of Algorand is under a synchronous network setting, where messages are delivered within a step.

Forensics. Algorand is designed for $n = 3t + 1$ players of which t are Byzantine. If the Byzantine players are beyond t, they can deviate from the protocol and create a safety violation. Unlike HotStuff or Streamlet, there exists a safety attack that does not leave cryptographic evidence and malicious actors are not held accountable. For instance, in the second step, if all honest players agree on $b = 0$ at the start of the step, but more than $1/3$ fraction of players are Byzantine and do not send any vote, then every honest player won't receive $2/3$ fraction of votes for $b = 0$, and they will switch their local value to $b = 1$ by the end of this step. This can result in a safety violation. Nevertheless, since the Byzantine players don't send any message at all, there is no cryptographic evidence to hold them accountable. This statement holds for Algorand with or without committee elections and player replaceability. Whether there is a BFT protocol that is efficient and has player replaceability (so that it can be used as a core consensus engine inside a PoS permissionless blockchain) with strong forensic support is an open question.

References

[1] Silvio Micali. Byzantine agreement, made trivial, 2018.

[2] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.

[3] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. Bft protocol forensics, 2020.