CHAPTER 8

Blockchain Age Protocols

This chapter covers blockchain age protocols. Some novel protocols and some variants of classical blockchain consensus protocols were discussed in Chapter 7. We start with Ethereum and finish this chapter with a discussion on Solana. Along the way, we will cover in detail the characteristics, strengths, weaknesses, properties, and inner workings of major consensus protocols used in platforms such as Cosmos, Ethereum 2.0, and Polkadot.

We already covered proof of work in detail in Chapter 5. So, I will not repeat that here; however, Ethereum's PoW will be discussed in this chapter.

Introduction

Consensus protocols are at the core of any blockchain. A new class of consensus protocols emerged with Bitcoin. Therefore, we can categorize all consensus protocols for a blockchain that emerged with and after Bitcoin as "blockchain age consensus protocols."

The primary aim of a consensus protocol in a blockchain is to achieve an agreement on the state of the blockchain while preserving the safety and liveness of the system. The state generally refers to the value, history, and rules of the blockchain. An agreement on the canonical history of the blockchain is vital, and so is the agreement on the governing rules of the chain. Additionally, consensus on values (data) added to the chain is fundamentally critical.

Like traditional pre-blockchain protocols, safety and liveness are two key properties that should be fulfilled by a consensus protocol to ensure the consistency and progress of the blockchain.

Blockchain consensus protocols can be divided into two main categories: the probabilistic finality protocols and absolute finality protocols – in other words, probabilistic termination protocols and deterministic termination protocols. Probabilistic protocols are abundantly used in cryptocurrency public blockchains like Ethereum and Bitcoin. Deterministic protocols, usually from the BFT class of

I. Bashir, *Blockchain Consensus*, https://doi.org/10.1007/978-1-4842-8179-6_8

protocols, are commonly used in enterprise blockchains; however, they are also used in some public blockchains. While PBFT variants are more commonly used in enterprise blockchains, their usage in public chains is somewhat limited only to some public blockchains. For example, TowerBFT used in Solana is a deterministic finality consensus protocol. BFT-DPOS used in EOSIO is another example. Deterministic finality is also known as *forward security* where a guarantee is provided that a transaction once finalized will not be rolled back.

From the perspective of how the consensus algorithms work, blockchains or distributed ledgers are based on one or a combination of the following types of consensus algorithms:

- **PoW based**: Such as Nakamoto consensus in Bitcoin, which relies on solving a math puzzle using brute force.
- **Leader based**: Such as usual BFT protocols where a leader acts as a primary proposer of blocks/values.
- **Voting based**: Usually applicable in BFT protocols where a leader gathers votes from followers to finalize a decision. Also called "quorum based."
- **Virtual voting**: Usually, voting in BFT protocols is complex from a communication point of view, where each voter has to send and receive several messages to the leader. Virtual voting is a technique in the Hashgraph algorithm used in Hedera where votes are evaluated by looking at the local copies of the Hashgraph instead of complex communication with other nodes. This process eventually leads to achieving a Byzantine agreement.
- **Economy based**: Such as proof of stake mechanisms that rely on a stake bonded in the network.

After Bitcoin's inception, many blockchains emerged, and alternative PoW algorithms were introduced, for example, Litecoin. As PoW consumes excessive amounts of energy, the community felt very early that alternatives that are not excessively energy consuming need to be designed. In the wake of introducing less energy-consuming protocols, developers introduced proof of stake. With PoS, the sustainable public blockchain networks have become possible to build. There are, however, some challenges and caveats. After going through the mechanics of how PoS works, we will discuss these limitations.

Proof of Stake

Even though Bitcoin's PoW has proven to be a resilient and robust protocol, it has several limitations:

- Excessive energy consumption
- Slow rate of block generation
- Becoming centralized due to the requirement of specialized hardware and large mining pools
- Probabilistic finality which is not suitable for most of the applications
- Not a perfect consensus algorithm, some attacks exist, for example, Goldfinger attack, 51% attack
- Barrier to entry getting higher as a special hardware requirement to mine
- Not scalable enough to support usual high throughput applications

Significant research has been ongoing to address the abovementioned weaknesses. Especially, high energy consumption from the limitations mentioned earlier led to the development of alternatives. Proof of stake is one such alternative.

Proof of stake first appeared in Peercoin in 2012. Later, many blockchains adopted this mechanism, such as EOS, NxT, Steem, Tezos, and Cardano. In addition, Ethereum, with its Serenity release, will soon transition to a PoS-based consensus mechanism. Proof of stake is also called virtual mining. This is so because in PoS instead of requiring miners to allocate compute resources to solve a puzzle, the right to produce the next block is decided on the basis of what value the miner possesses. This valuable possession can be anything of value (usually coins) that aligns with the interest of the network. PoW's motto was one CPU = one vote, whereas we can think of PoS as one coin = one vote.

The next proposer is usually elected randomly. Proposers are incentivized either with transaction fees or block rewards. Like PoW, control over the majority of the network in the form of controlling a large portion of the stake is required to attack and control the network.

PoS protocols usually select stakeholders and grant suitable rights based on their staked assets. The stake calculation is application specific but is typically based on the total balance, deposited value, or voting between the validators. Once the stake is calculated and a stakeholder is selected as the block proposer, the block proposed by the proposer is readily accepted. The higher the stake, the better the chances of winning the right to propose the next block.

A general scheme of a PoS scheme is shown in Figure 8-1.

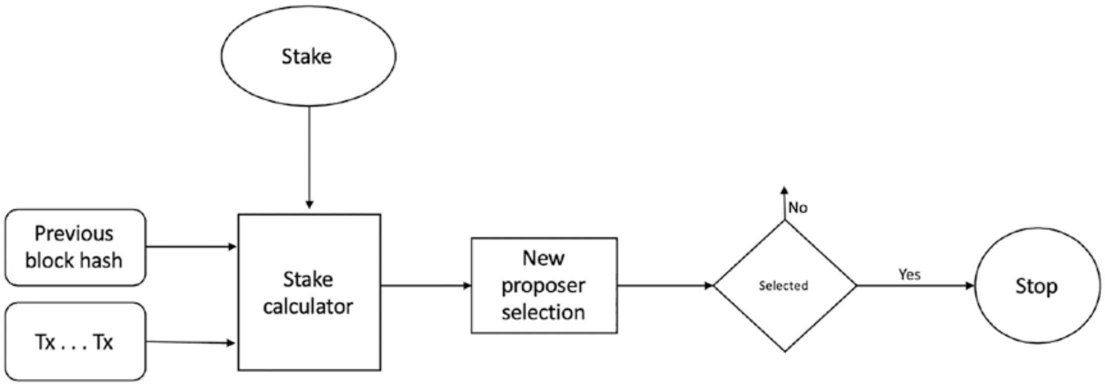


Figure 8-1. *Proof of stake scheme*

As shown in Figure 8-1, PoS uses a stake calculator function to calculate the amount of staked funds, and, based on that, it selects a new proposer.

The next proposer is usually elected randomly. Proposers are incentivized either with transaction fees or block rewards. Control over most of the network by having a large portion of the stake is needed to attack the network.

There is some element of randomness introduced in the selection process to ensure fairness and decentralization. Other factors in electing a proposer include the age of the tokens, which takes into account for how long the staked tokens have been unspent; the longer the tokens have been unspent, the better the chances to be elected.

There are several types of PoS:

- Chain-based PoS
- BFT-based PoS
- Committee-based PoS
- Delegated proof of stake
- Liquid proof of stake

Chain-Based PoS

This scheme is the first alternative proposed to PoW. It was used first in Peercoin in 2012. This mechanism is like PoW; however, the block generation method is changed, which finalizes blocks in two steps:

- Pick transactions from the memory pool and create a candidate block.
- Set up a clock with a constant tick interval. At each clock tick, check whether the hash of the block header concatenated with the clock time is less than the product of the target value and the stake value. We can show this simple formula as follows:

$$Hash\left(B_h \,\|\, clock\ \ time\right) < target \times stake\ value$$

The stake value depends on how the algorithm works. In some chains, it is proportional to the amount of stake. In others, it is based on the amount of time the participant has held the stake. The target is the mining difficulty per unit of the value of the stake.

This mechanism uses hashing puzzles, as in PoW. But, instead of competing to solve the hashing puzzle by consuming high energy and using specialized hardware, the hashing puzzle in PoS is solved only once at regular clock intervals. A hashing puzzle becomes proportionally easier to solve if the stake value of the miner is high. This contrasts with PoW where repeated brute-force hashing is required to solve the math puzzle.

Committee-Based PoS

In this scheme, a group of stakeholders is chosen randomly, usually using a verifiable random function (VRF). The VRF produces a random set of stakeholders based on their stake and the current state of the blockchain. The chosen group of stakeholders becomes responsible for proposing blocks in sequential order.

A general scheme is described as follows:

- A validator joins the network and deposits a stake.
- Participate in the committee election process and keep checking its turns.
- When it's the turn, collect transaction, generate block, append the new block in the chain, and finally broadcast the block.
- At the other receiver nodes, verify the block; if valid, append the block into the blockchain and gossip the block to others.

The committee election produces a pseudorandom sequence of turns for validators to produce blocks. Ouroboros Praos and BABE are common examples of committee-based PoS.

BFT-Based PoS

In this scheme, the blocks are generated using a proof of stake mechanism where a block proposer is chosen based on the proof of stake which proposes new blocks. The proposer is elected based on the stake deposited in the system. The chance of being chosen is proportional to the amount of stake deposited in the system. The proposer generates a block and appends it to a temporary pool of blocks from which the BFT protocol finalizes one block.

A general scheme works as follows:

- Elect a block proposed based on the PoS mechanism, proportional to a stake.
- Proposer: Propose a new block, add to a temporary block pool, and broadcast the new block.
- Receiver: When other nodes receive this, they validate the block and, if valid, add to the local temporary block pool.
- During the consensus epoch
    - Run BFT consensus to finalize a valid (most voted) block.
    - Add the most voted valid block to the main blockchain.
    - Remove other blocks from the temporary block pool.

Tendermint in Cosmos is one example where the validator is chosen based on a stake, and the rest of the protocol works on BFT principles. Other examples include Casper FFG.

BFT-based PoS is fault tolerant as long as two-thirds of validators remain honest. Also, blocks are finalized immediately.

Delegated PoS

DPoS works like proof of stake, but a critical difference is a voting and delegation mechanism which incentivizes users to secure the network. DPoS limits the size of the chosen consensus committee, which reduces the communication complexity in the protocol. The consensus committee is composed of so-called delegates elected by a delegation mechanism. The process works by stakeholders voting for delegates by using their stake. Delegates (also called witnesses) are identifiable, and voters know who they are, thus reducing the delegates' chance of misbehavior. Also, a reputation-based mechanism can be implemented, allowing delegates to earn a reputation based on the services they offer and their behavior on the network. Delegates can represent themselves for earning more votes. Delegates who get the most votes become members of the consensus committee or group. Usually, a BFT-style protocol runs between the members of the chosen consensus committee to produce and finalize blocks. Each member can take a round-robin fashion to propose the next block, but this activity remains within the elected consensus committee. Delegates earn incentives to produce blocks. Again, under the BFT assumptions, the protocol within the consensus committee can tolerate f faults in a 3f+1 member group. In other words, it can tolerate one-third or 33% of delegates being faulty. This protocol provides instant finality and incentives in proportion to the stake of the stakeholders. As network-wide consensus is not required and only a smaller group of delegators oversee making decisions, the efficiency increases significantly. Delegated PoS is implemented in EOS, Lisk, Tron, and quite a few other chains.

Liquid PoS

LPoS is a variant of DPoS. Token holders delegate their validation rights to validators without requiring transferring the ownership of the tokens. There exists a delegation market where delegates compete to become the chosen validator. Here, the competition is primarily on fees, services offered, reputation, payout frequency, and possibly other factors. Any misbehavior such as charging high fees by a validator is detectable quickly and will be penalized accordingly. Token holders are also free to move to any other validator. LPoS supports a dynamic number of validators as compared to DPoS's fixed validator set. Token holders are also allowed to become validators themselves by self-electing. Token holders with small amount can delegate to larger amount holders. Also, a number of small token holders can form a syndicate. Such "liquid" protocol allows much flexibility as compared to other PoS protocols and helps to thwart creation of lobbies to become a fixed validator set. LPoS is used in the Tezos blockchain.

There are some attacks against PoS, such as the nothing-at-stake attack, long-range attack, and stake grinding attack. We explain these attacks as follows.

Attacks

PoS suffers generally from a costless simulation problem where an adversary can simulate any history of the chain without incurring any additional cost, as opposed to PoW where the cost is computational power. This no-cost block generation is the basis of many attacks in PoS.

Nothing-at-Stake Problem

The nothing-at-stake or double bet problem occurs when multiple forks occur. An attacker can generate a block on top of each fork without any additional cost. To solve this problem, economic penalties are introduced in protocols that prevent attackers from launching this attack. If a significant number of nodes do this, then an attacker holding even less than 50% of tokens can launch a double-spend attack.

Long-Range Attacks

Long-range attacks exist due to weak subjectivity and costless simulation. Long-range attacks are also possible because of costless simulation where an adversary creates a new branch starting from the genesis block with the aim to take over the main good chain, once the bad chain becomes longer than the real main chain. This can create an alternate history which is detrimental to the blockchain.

A weak subjectivity problem affects new nodes and the nodes which were offline for a long time and rejoined the network. As nodes are not synchronized and there are usually multiple forks available in the network, these nodes are unable to differentiate between which node is correct and which one is malicious; they may as well accept a malicious fork as valid.

Other Attacks

Liveness denial is another attack that PoS can suffer from. In this attack, some or all validators collectively decide to stop validating the blocks, resulting in halting block production. Penalizing such activities by the protocol can prevent these types of attacks.

A **selfish mining** or block withholding attack occurs when an adversary mines their own chain offline. Once the chain is at a desired length, the adversary releases this chain to the network with the expectation that the bad chain will take over the main good chain. It can cause disruption on the network as it can result in causing honest validators to waste resources.

A **grinding attack** on PoS occurs if a slot leader election process is not random. If no randomness is introduced in this process, then a slot leader can increase the frequency of its own election again and again, which can result in censorship or disproportionate rewards. An easy way to solve this is to use some good random selection process, usually based on verifiable random functions (VRFs).

Next, we discuss Ethereum's proof of work – Ethash.

Ethereum's Proof of Work

We discussed PoW for Bitcoin in detail in Chapter 5. In this section, we'll see how ETHASH works, the PoW used in Ethereum.

Ethash is the evolved form of the Dagger-Hashimoto algorithm. The key idea behind mining is to find a nonce (an arbitrary random number), which, once concatenated with the block header and hashed, results in a lower number than the current network difficulty level. Initially, the difficulty was low when Ethereum was new, and even CPU and single GPU mining was profitable to a certain extent, but that is no longer the case. So now, only pooled mining or large GPU mining farms are used for profitable mining purposes.

Ethash is a memory-hard algorithm, making it challenging to implement on specialized hardware due to the requirement of large and fast memories on ASICS, which is generally not practical.

Note that for quite some time, this memory hardness of Ethash prevented the development of ASICs, but now various ASIC miners are available for Ethereum mining.

This algorithm requires subsets of a fixed resource called a directed acyclic graph (DAG) to be chosen, depending on the nonce and block headers.

DAG is a large, pseudorandomly generated dataset. This graph is represented as a matrix in the DAG file created during the Ethereum mining process. The Ethash algorithm has the DAG as a two-dimensional array of 32-bit unsigned integers. Mining only starts when DAG is fully created the first time a mining node starts.

This DAG is used as a seed by the Ethash algorithm. The Ethash algorithm requires a DAG file to work. A DAG file is generated every epoch, 30,000 blocks. DAG grows linearly as the chain size grows.

The protocol works as follows:

- The header from the latest block and a 32-bit random nonce are combined using the Keccak-256 hash function.
- This produces a 128-bit structure called mix which determines which data, that is, a 128-byte page, to select from the DAG.
- Once the data is fetched from the DAG, it is "mixed" with the mix to produce the next mix, which is then used to fetch data from the DAG and subsequently mixed again. This process repeats 64 times.
- Eventually, the 64th mix is run through a digest function to produce a 32-byte sequence called the mix digest.
- This sequence is compared with the difficulty target. If it is less than the difficulty target, the nonce is valid, and the PoW is solved. As a result, the block is mined. If not, then the algorithm repeats with a new nonce.

We can visualize this process in Figure 8-2.

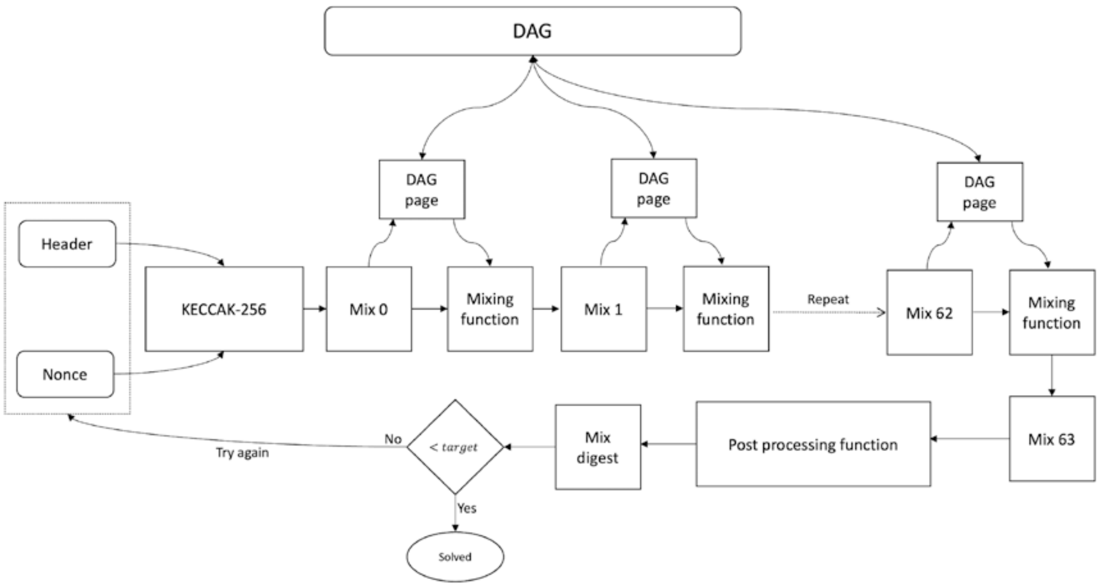


Figure 8-2. *Ethash process*

Ethash has several objectives:

- The algorithm consumes almost all available memory access bandwidth, which is an ASIC resistance measure.
- Mining with Ethash using GPUs is easier to perform.
- Light clients can verify mining rounds much efficiently and should be able to become operational quickly.
- The algorithm runs prohibitively slow on light clients as they are not expected to mine.

As the Ethereum execution layer (formerly Eth1) advances toward the consensus layer (formerly Ethereum 2), this PoW will eventually phase out. When the current EVM chain is docked into the beacon chain, that is, the so-called “the merge” happens, Casper FFG will run on top of PoW. Eventually, however, Casper CBC, the pure PoS algorithm, will finally take over.

Also, with ice age activation, the PoW will become almost impossible to mine due to the extreme difficulty level induced by the “ice age,” and users will have no choice but to switch to PoS.

Solana

Solana is a layer 1 blockchain with smart contract support introduced in 2018. Developers of Solana aimed for speed, security, scalability, and decentralization. At the time of writing, it is in Beta, and it is growing in popularity quickly. Though it is an operational network with production systems running on it, there are some technical issues which are being addressed.

The ledger is a verifiable delay function where time is a data structure, that is, data is time. It potentially supports millions of nodes and utilizes GPUs for acceleration. SOL coin is the native token on the platform used for governance and incentivization. The main innovations include the following.

Proof of history (PoH) enables ordering of events using a data structure–based cryptographic clock instead of an external source of time, which then leads to consensus.

TowerBFT is a protocol for consensus derived from PBFT. Note that PoH is not a consensus protocol, it is simply a mechanism to enable ordering of events using a data structure–based clock. A consensus mechanism is still needed to enable nodes to vote on a correct branch of the ledger.

Turbine is another innovation which enables block propagation in small chunks called shreds, which helps to achieve speed and efficiency. There are no memory pools in Solana as transactions are processed so fast that memory pools do not form. This mechanism has been named the Gulf stream.

Solana supports parallel execution of smart contracts, which again results in efficiency gains.

Transactions are validated in an optimized fashion using pipelining in the so-called transaction processing unit residing within validators.

Cloud break is a name given to the database which is horizontally scalable. Finally, archivers or replicators are nodes which allow for distributed ledger storage, as in a high throughput system such as Solana data storage can become a bottleneck. For this purpose, archivers are used which are incentivized to store data.

As our main focus is consensus algorithms, I will leave the introduction to blockchains here and move on to discussing the actual consensus and relevant mechanisms in Solana.

Solana uses proof of stake and TowerBFT consensus algorithms. One of the key innovations in Solana is proof of history, which is not a consensus algorithm but allows to create a self-consistent record of events proving that some event occurred before and after some point in time. This then leads to consensus. It results in reducing the message complexity in a BFT protocol where effectively communication is replaced by local computations. This immediately results in high throughput and subsecond finality times. It has been long known in the distributed systems research community that if somehow communication can be replaced with local computation, then significant performance can be achieved, and many clock synchronization protocols emerged as a result. However, generally all of these relied on an external source of time either via atomic clocks or GPS and then use an NTP type of protocol to synchronize. PoH is a proof for cryptographically proving order and passage of time between events without relying on an external source of time.

Proof of History

As discussed in Chapter 1, time in distributed systems is crucial. If time is synchronized among processes, that is, a synchronized clock is available in a distributed network, then communication can be reduced, which results in improved performance. A node can deduce information from past events instead of asking another node repeatedly about some information. For example, with the availability of a global clock where all nodes are synchronized, the system can establish a notion of the system-wide history of events. For example, a timestamp on an event can inform a node when this event occurred in reference to the globally synchronized time across the network, instead of asking a node again who produced that event when this event occurred.

Another application of a synchronized clock is that entities in the system can deduce if something has expired, for example, a timestamped security token can immediately tell a node how much time has elapsed since its creation. The node can infer if it is valid anymore or not and not something that occurred in the distant past, making this token no longer applicable and expired.

In replication protocols, clock synchronization also plays a crucial role. If nodes don't have clocks synchronized, that can lead to inconsistency because every node will have a different view of the order of events.

If the time is not synchronized among nodes, the system cannot establish a global notion of time and history. It is usually possible in practical systems using an NTP protocol. We discussed this in Chapter 2 before in the context of time and order in distributed systems.

So far, we have established that synchronized time is indeed a valuable construct in distributed systems for performance gains. In other words, if we can somehow replace communication with local computation, then we can gain tremendous efficiency.

Also, synchrony and time together solve consensus easily. Safety and liveness, the two fundamental requirements, are easy to implement with a trusted clock and synchronous network. However, networks are empirically asynchronous. We also know that a trusted synchronized clock in distributed networks is difficult to maintain. Blockchains and distributed systems are characterized by no clocks, which make them slow due to inherent asynchrony and the need for complex message passing for ordering of events and agreement.

On the other hand, a reliable, trusted clock makes network synchronization much simpler and quicker, which leads to very fast networks. Solana's PoH is a solution where the system can keep time reliably between nontrusting computers. In short, PoH enables clocks in clockless blockchains.

Solana's PoH is a way to establish the history and provide that global notion of synchronized time among nodes in a distributed network. The key innovation here is that it does not use any external source of time and synchronize nodes using that, for example, via an NTP protocol; instead, it uses a cryptographic proof to show that some time has passed, and other nodes directly accept this history of events due to cryptographic guarantees. So instead of relying on a source of global time, this mechanism is built into validators that generate a sequence of events with a proof when an event has occurred. The following describes how it works.

In a blockchain network, the right of adding a new block is won after solving a puzzle, that is, PoW, which takes a long time. Although this mechanism is secure and thwarts Sybil attacks (as we saw in Chapter 5), it is slow. If BFT-style consensus is used, the leader validator that proposes a block only gets to commit after at least two sequential phases, which is also time-consuming even under a normal environment. In case of failures, it can ever further slow down with new leader selection (election) and view changes. What if somehow there is a deterministic leader election algorithm that can select leaders in quick successions, and each leader quickly proposes, and then the algorithm moves to the next leader and so on? All this without going through complex leader election, acknowledgment from other nodes, and running multiple phases to reach consensus. The problem here is that it's quick to create a deterministic algorithm that can select the next leader, but how do I ensure that what they propose is correct and that selected leaders are not malicious and will not censor transactions or exhibit other malicious behaviors?

This is where PoH comes in. In Solana, one leader at a time processes transactions and updates the state. Other validators read the state and send votes to the leader to confirm them. This activity is split into very short successive sessions where one leader after another performs this. It can be thought of as if the ledger is split into small intervals. These small intervals are of 400ms each. The leader rotation schedule is predetermined and deterministic based on several factors such as the stake and behavior of previous transactions. But how can we ensure that the leader rotation is done at the right time and does not skip the leader's turn?

In PoH, the passage of time is proven by creating a sequence of these hashes, as shown in Figure 8-3.

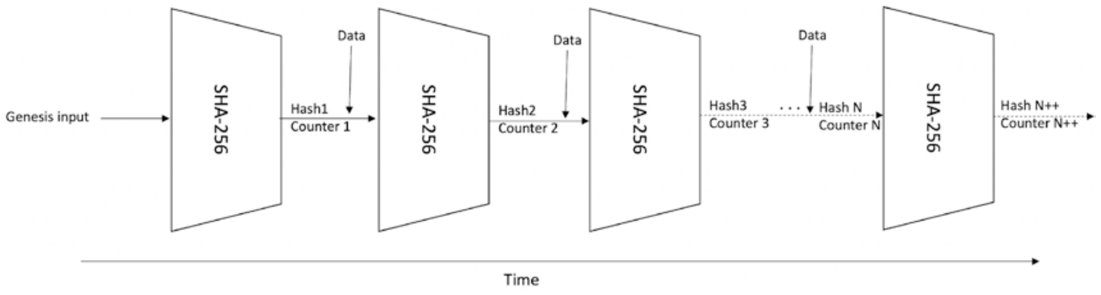


Figure 8-3. *Solana proof of history sequence*

In Figure 8-3, a sequence of hash operations is shown. The genesis input (shown at the left in the diagram) is first provided to the hash function. In the next iteration, the output of the previous hash function is used as an input to the hash function, and this process continues indefinitely. This sequence is generated using the SHA-256 function on a single core. This process cannot parallelize it because the output of the previous hash function can only be known if and only if the hash function has processed the previous input. It is assumed that the functions are cryptographic hash functions that are preimage resistant. Therefore, this is a purely sequential function. However, this sequence can be verified in parallel using multicore GPUs. As all the inputs and outputs are available, it becomes just a matter of verifying each output, which GPUs can do in parallel. This property makes this sequence a verifiable delay function (VDF) because the time taken (i.e., delay) in generating the hash sequence can be verified using quick parallel verification. However, there is some debate between cryptographic VDFs introduced by researchers at Stanford and hardware VDF introduced by Solana researchers. See the reference in the bibliography.

We can then sample this sequence at regular intervals to provide a notion of the passage of time. This is so because hash generation takes some CPU time (roughly 1.75 cycles for SHA-256 instruction on an Intel or AMD CPU), and this process is purely sequential; we can infer from looking at this sequence, that since the first hash is generated, up to a later hash in the sequence, some time has passed. If we can also add some data with the input hash to the hash function, then we can also deduce that this data must have existed before the next hash and after the previous hash. This sequence of hashes thus becomes a proof of history, proving cryptographically that some event, let's say event e, occurred before event f and after event d.

It is a sequential process that runs SHA-256 repeatedly and continuously, using its previous output as its input. It periodically records a counter for each output sample, for example, every one second, and current state (hash output), which acts like clock ticks. Looking at this structure of sampled hashes at regular intervals, we can infer that some time has passed. It is impossible to parallelize because the previous output is the input for the next iteration. For example, we can say time has passed between counter 1 and counter N (Figure 8-3), where time is the SHA-256 counter. We can approximate real time from this count. We can also associate some data, which we can append to the input of the hash function; once hashed, we can be sure that data must have existed before the hash is generated. This structure can only be generated in sequence; however, we can verify it in parallel. For example, if 4000 samples took 40 seconds to produce, it will take only 1 second to verify the entire data structure with a 4000 core GPU.

The key idea is that PoH transactional throughput is separated from consensus, which is key to scaling. Note that the order of events generated, that is, the sequence, is not globally unique. Therefore, a consensus mechanism is needed to ascertain the true chain, as anyone can generate an alternate history.

Proof of history is a cryptographically proven way of saying that time has elapsed. It can be seen as an application-specific verifiable delay function. It encodes the passage of time as data using SHA-256 hashing to hash the incoming events and transactions. It produces a unique hash and count of each event, which produces a verifiable ordering of events as a function of time. This means that time and ordering of events can be agreed without waiting to hear from other nodes – in other words, no weak subjectivity where nodes must rely on other nodes to determine the current state of the system. This results in high throughput, because the information that is usually required to be provided by other nodes is already there in the sequence generated by the PoH mechanism and is cryptographically verifiable, ensuring integrity. This means that a global order of events can be enforced without going through a communication-wise complex agreement protocol or trusting an external source of time for clock synchronization. In summary, instead of trusting the timestamp, proof of history allows to create a historical record proving that an event e occurred at a particular point in time t, before another event f and after an event d.

Using PoH, leadership can be switched without needing to communicate with other nodes, which results in increased block production frequency. PoH results in high node scalability and low communication complexity as compared to BFT-style protocols with high communication complexity and limited node capacity.

It provides global read consistency and cryptographically verifiable passage of time between two events. With PoH, nodes can trust the ordering and timing of events, even before the consensus stage is reached. In other words, it's a clock before consensus approach. Consensus then simply works by voting on different branches where nodes vote on a branch that they believe is the main chain. Over time, by keep voting on the chain they first voted on, and by voting on any other branch, they earn rewards and eventually the other branches orphan.

TowerBFT is a variant of PBFT. It is basically a fork selection and voting algorithm. It is used to vote on the chains produced by PoH to select the true canonical chain. It is less communication-wise complex because PoH has already provided an order, and now the decision is only required on the choice of canonical chain. PoH provides timing of events before consensus initiates, and TowerBFT is then used for voting on the canonical chain.

TowerBFT is BFT in the sense that once two-thirds of validators have voted on a chain (hash), then it cannot be rolled back. Validators vote on a PoH hash for two reasons: first, the ledger is valid up until that hash, that is, a point in time, and, second, to support for a fork at a given height as many forks can exist at a given height.

Furthermore, PoS in Solana is used for economics and governance to control slashing, inflation, supply, and penalties.

Tendermint

Tendermint is inspired by the DLS protocol that we covered in Chapter 6 and was originally introduced in the DLS paper. It can be seen as a variant of PBFT too with similarities in the phases.

The Tendermint protocol works in rounds. In each round, an elected leader proposes the next block. In Tendermint, the view change process is part of the normal operation. This concept is different from PBFT, where a view change only occurs in the event of a suspected faulty leader. Tendermint works similarly to PBFT, where three phases are required to achieve consensus. A key innovation in Tendermint is the design of a new termination mechanism. Unlike other PBFT-like protocols, Tendermint has developed a more straightforward mechanism like a PBFT-style normal operation. Instead of having two subprotocols for normal mode and view change mode (recovery in case of a faulty leader), Tendermint terminates without additional communication costs.

Tendermint works under some assumptions about the operating environment, which we describe next:

Processes: A process is a participant on the network. Processes are expected to be honest, but they can turn faulty. Each process has a voting power that serves as a confirmation to the leader. Processes can connect loosely or with their immediate subset of processes/nodes. They are not necessarily connected directly. Processes have a local timer that they use to measure timeout.

Network model: The network is a message-passing network where a gossip protocol is used for communication between processes. The standard BFT assumption of $n \geq 3f + 1$ applies here, which means that the protocol operates correctly only if the number of nodes in the network stays more than 3F, where F is the number of faulty nodes and N represents the total number of nodes in the network. In practice, this means that there must be at least four nodes in a network to tolerate Byzantine faults.

Timing assumptions: Tendermint assumes a partially synchronous network. There is an unknown bound on the communication delay, but it only applies after an unknown instance of time called global stabilization time or GST.

Security and cryptography: The security assumption in the system is that the public key cryptography used is secure. Also, the impersonation or spoofing of identities is not possible. All messages on the network are authenticated and verified via digital signatures. The protocol ignores any messages with an invalid digital signature.

State machine replication: SMR is used to achieve replication among the nodes. SMR ensures that all processes on the network receive and process the same sequence of requests. In addition, the agreement and order provide that the sequence in which the nodes have received requests is the same on all nodes. Both requirements ensure the total order in the system. The protocol only accepts valid transactions.

Tendermint solves consensus by fulfilling the properties listed as follows:

- **Agreement**: No two correct processes decide on different values.
- **Termination**: All correct processes eventually decide on a value.
- **Validity**: A decided-upon value is valid if it satisfies an application specific predefined predicate denoted valid( ).

State transition at processes in Tendermint depends on the messages received and timeouts. The timeout mechanism guarantees liveness and prevents indefinite waiting. Here, the assumption is that eventually, after some period of asynchrony, there will be a synchronous communication period during which all processes can communicate in a timely fashion, ensuring that processes eventually decide on a value.

A Tendermint protocol has three types of messages: proposal, pre-vote, and pre-commit. These messages can be viewed as equivalent to the PBFT protocol's PRE-PREPARE, PREPARE, and COMMIT messages:

- **Proposal:** This message is used by the leader of the current round to propose a value or block.
- **Pre-vote:** This message is used to vote on a proposed value.
- **Pre-commit:** This message is also used to vote on a proposed value.

Only the proposal message contains the original value. The other two messages, pre-vote and pre-commit, use a value identifier representing the initially proposed value.

There are three timeouts in the protocol, corresponding to each message type:

- Timeout-propose
- Timeout-prevote
- Timeout-precommit

These timeouts prevent the algorithm from waiting indefinitely for certain conditions to be met. They also ensure that processes make progress through the rounds. A mechanism to increase timeout with every new round assures that after reaching GST, the communication between correct processes eventually becomes reliable, nodes can reach a decision, and protocol terminates.

All processes maintain some necessary variables in the protocol:

- **Step:** This variable holds the current state of the Tendermint state machine in the current round.
- **lockedValue:** This variable stores the most recent value (concerning the round number) for which a pre-commit message has been sent.
- **lockedRound:** This variable holds information about the last round where the process sent a non-nil pre-commit message which implies that this is the round where a possible decision value has been locked. This means that if a proposal message and corresponding 2F + 1 messages have been received for a value in a round, then, due to the reason that 2F + 1 pre-votes have already been accepted for this value, this is a possible decision value.
- **validValue:** The role of the validValue variable is to store the most recent possible decision value.
- **validRound:** The validRound variable is the last round in which validValue was updated.
- **Height:** Stores the current consensus instance.
- Current round number
- An array of decisions

Tendermint proceeds in rounds. Each round contains three phases: propose, pre-vote, pre-commit. The algorithm works as follows:

- Every round starts with a proposal value proposed by a proposer. The proposer proposes a new value at the start of the first round for each height.

- Any subsequent rounds will only have a proposer proposing a new value if there is no valid value already present, that is, null. Otherwise, the validValue, the possible decision value, is proposed, already locked from a previous round. The proposal message also includes a value denoting the last valid round in which there was a valid value updated.
- A correct process accepts the proposal only if
    - The proposed value is valid.
    - The process has not locked on a value.
    - Or the process has a value locked.
- The correct process accepts the proposal and sends a pre-vote message if the preceding conditions meet.
- If the conditions do not meet, the process will send a pre-vote message with a nil value.
- A timeout mechanism associated with the proposal phase triggers timeout if a process has not sent a pre-vote message in the current round or the timer expires in the proposal stage.
- If a correct process receives a proposal message with a valid value and 2F + 1 pre-vote messages, it sends the pre-commit message.
- Otherwise, it sends out a nil pre-commit.
- A timeout mechanism associated with the pre-commit will initialize if the associated timer expires or if the process has not sent a pre-commit message after receiving a proposal message and 2F + 1 pre-commit messages.
- A correct process decides on a value if it has received the proposal message in some round and 2F + 1 pre-commit messages for the ID of the proposed value.
- This step also has an associated timeout mechanism, ensuring that the processor does not wait indefinitely to receive 2F + 1 messages. If the timer expires before the processor can decide, the processor starts the next round.

- When a processor eventually decides, it triggers the next consensus instance for the following block proposal, and the entire cycle of a proposal, pre-vote, and pre-commit starts again.

The protocol can be simply depicted as a recurring sequence of proposal ➤ pre-vote ➤ pre-commit, and after every commit, a new height is achieved and a new round starts, as shown in Figure 8-4.

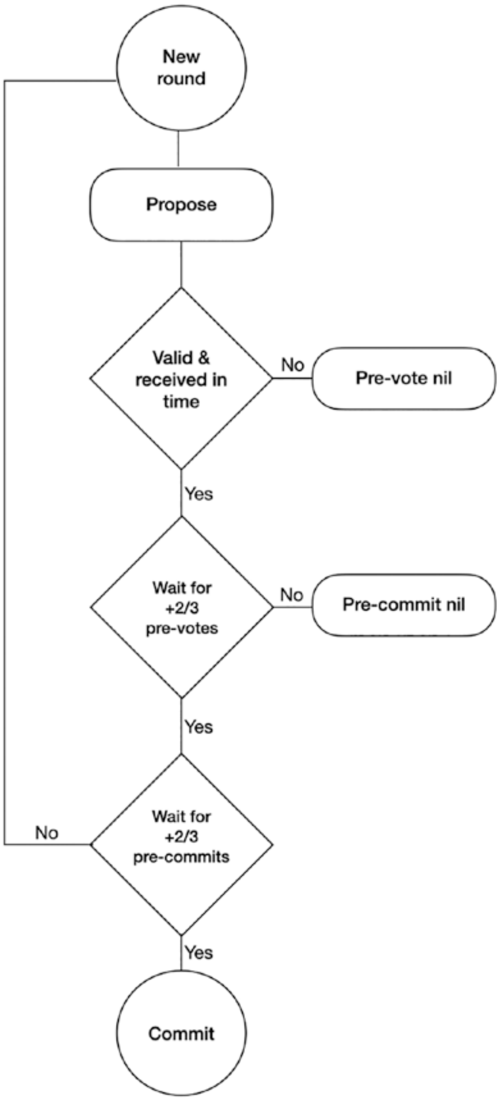


Figure 8-4. *Tendermint flow – single run*

Tendermint introduced a new termination mechanism. There are two variables, namely, `validValue` and `validRound`, used by the proposal message. Both are updated by a correct process when receiving a valid proposal message and subsequent corresponding 2f + 1 pre-vote messages.

This termination process benefits from the gossip protocol and synchrony assumptions. For example, suppose a correct process has locked a value in a round. Then all other correct processes will update their `validValue` and `validRound` variables with the locked values by the end of the very round during which they were locked. The fundamental presumption is that a gossip protocol will propagate them to other nodes within the same round once a correct processor has locked these values. Each processor will know the locked value and round, that is, the valid values. Now, when the next proposal is made, the same locked values will be picked up by the proposer, which has already been locked due to the valid proposal and corresponding $2f + 1$ pre-vote messages. This way, it can be ensured that the value that processes eventually decide upon is acceptable as specified by the validity condition.

This completes our discussion on the Tendermint protocol. Next, we explore HotStuff, which improves on several limitations in previous PBFT and its variant protocols.

HotStuff

HotStuff is a BFT protocol for state machine replication. Several innovations make it a better protocol than traditional PBFT. However, like PBFT, it works under partial synchrony in a message-passing network with minimum $n = 3f + 1$ and relies on a leader-based primary backup approach. It utilizes reliable and authenticated communication links. HotStuff makes use of threshold signatures where all nodes use a single public key, but each replica uses a unique private key. The use of threshold signatures results in reduced communication complexity.

HotStuff introduced some innovations which we introduce as follows.

Linear View Change

A view change in a HotStuff protocol requires only $O(n)$ messages. It is part of the normal run instead of a separate subprotocol. In a worst-case scenario where leaders fail successively, the communication cost increases to $O(n^2)$, quadratic. Instead of a stable leader like PBFT, a leader rotation in HotStuff occurs every three rounds even if the leader doesn’t fail.

In simpler words, quadratic complexity means that the algorithm's performance is proportional to the squared size of the input.

In a linear view change, after GST, any honest selected leader sends only $O(n)$ authenticators to drive a decision, including the case where a leader fails and a new one is elected. So even in the worst case where leaders fail one after another, the communication cost to reach consensus after GST is $O(n^2)$.

Optimistic Responsiveness

Optimistic responsiveness allows any correct leader after GST to only need the first $n - f$ responses to ensure progress instead of waiting for $n - f$ from every replica. This means that it operates at network speed instead of waiting unnecessarily for more messages from other nodes and move to the next phase.

Chain Quality

This property provides fairness and liveness in the system by allowing frequent leader rotation.

Hidden Lock

It also solves the hidden lock problem. A "hidden lock" problem occurs when a leader validator does not wait for the expiration time of a round. The highest lock may not get to the leader if we rely only on receiving $n - f$ messages. The highest locked value may be held in another replica from which the leader did not wait to get a response, thus resulting in a situation where the leader is unaware of the highest locked value. If a leader then proposes a lower lock value and some other nodes already have a higher value locked, this can lead to liveness issues. The nodes will wait for a higher lock or the same lock reply, but the leader is unaware of the highest lock value and will keep sending a lower lock value, resulting in a race condition and liveness violation.

HotStuff has solved this problem by adding the precursor lock round before the actual lock round. The insight here is that if $2f + 1$ nodes accept the precursor lock, the leader will get a response from them and learn the highest locked value. So now the leader doesn't have to wait for Δ (delta – an upper bound on a message delivery delay) time and can learn the highest lock with $n - f$ responses.

Pacemaker

HotStuff innovatively separates the safety and liveness mechanisms. Safety is ensured through voting and commit rules for participants in the network. On the other hand, liveness is the responsibility of a separate module, called pacemaker, which ensures a new, correct, and unique leader is elected. Furthermore, pacemaker guarantees progress after GST is reached. The first responsibility it has is to bring all honest replicas and a unique leader to a common height for a sufficiently long period. For synchronization, replicas keep increasing their timeouts gradually until progress is made. As we assume a partially synchronous model, this mechanism is likely to work. Also, the leader election process is based on a simple rotating coordinator paradigm, where a specific schedule, usually round-robin, is followed by replicas to select a new leader. Pacemaker also ensures that the leader chooses a proposal that replicas will accept.

Better Participant Organization Topology

A PBFT protocol organizes nodes in a clique (mesh topology), resulting in a quadratic message complexity, whereas HotStuff organizes nodes in a star topology. This setting enables the leader to send or collect messages directly to or from all other nodes, which results in reduced message complexity. In simpler words, it uses a "one-to-all" communication pattern.

If a leader is responsible for all this processing, the problem of high load on a single leader can arise, which can slow down the network.

Now think of a scenario where the leader gets corrupted or compromised. The standard BFT tolerance guarantees address such situations. If a leader proposes a malicious block and is suspected faulty, it will be rejected by other correct nodes, and the protocol will choose a new leader. This scenario can temporarily slow down the network until a new honest leader takes over. If majority of the network is honest, a correct leader will eventually take over and propose a valid block. Also, for added security, the leader usually is frequently rotated between validators every few rounds, which can offset any malicious attacks targeting the leader. This property ensures fairness, which helps to achieve chain quality.

PBFT consists of a normal and view change mode, where a view change triggers when a leader is suspected faulty. This approach provides a liveness guarantee but increases communication complexity. HotStuff handles this by combining the view change process with the normal mode. In PBFT, nodes wait for 2F+1 messages before the view change occurs, but in HotStuff, the view change can occur directly without invoking a separate subprotocol. Instead, checking the threshold of the messages to change the view becomes part of the normal view.

How It Works

HotStuff is composed of four phases: prepare, pre-commit, commit, and decide phases.

A quorum certificate (QC) is a data structure that represents a collection of signatures produced by $n - f$ nodes to indicate that a required threshold of messages has been achieved. In other words, a collection of votes from $n - f$ nodes is a QC.

Prepare

Once a new leader has accumulated new view messages from N – F nodes, the protocol starts with a new leader. The leader processes these messages to determine the latest branch in which the highest quorum certificate of PREPARE messages is present.

Pre-commit

As soon as a leader accumulates N – F prepare votes, it creates a quorum certificate called "prepare quorum certificate." The leader broadcasts this certificate to other nodes as a PRE-COMMIT message. When a node receives the PRE-COMMIT message, it responds with a pre-commit vote. The quorum certificate indicates that the required threshold of nodes has confirmed the request.

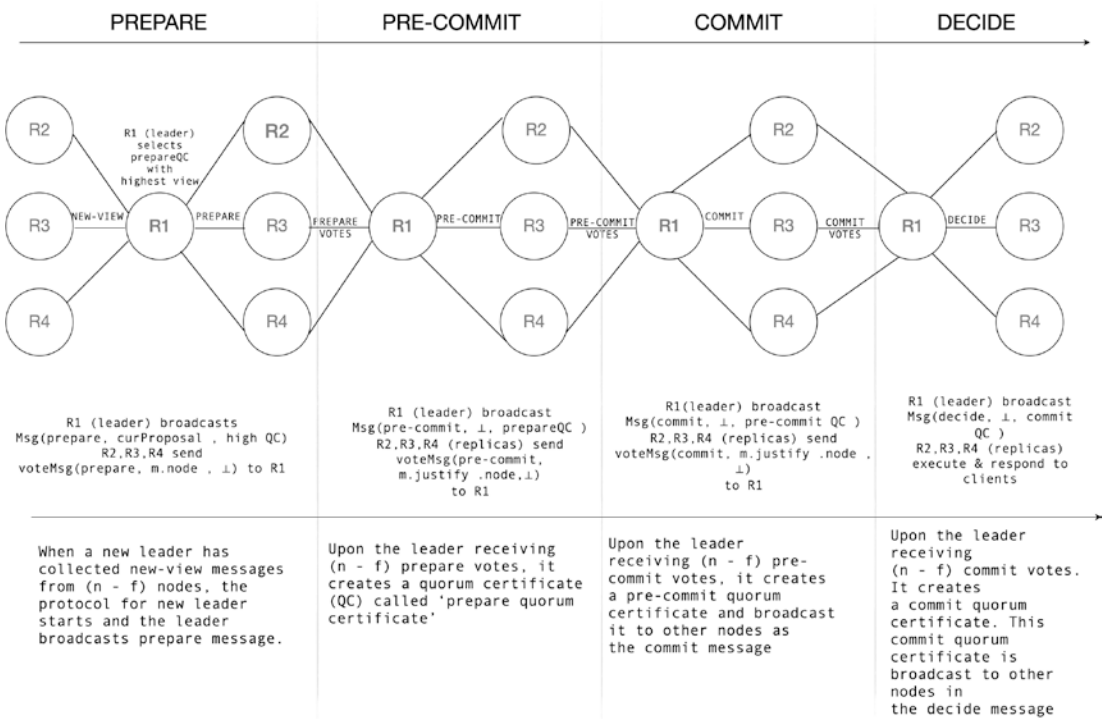
Commit

When the leader has accumulated N – F pre-commit votes, it creates a PRE-COMMIT quorum certificate and broadcasts it to other nodes as the COMMIT message. When nodes receive this COMMIT message, they respond with their commit vote. At this stage, nodes lock the PRE-COMMIT quorum certificate to ensure the safety of the algorithm even if a view change occurs.

Decide

When the leader receives N – F commit votes, it creates a COMMIT quorum certificate. Then, the leader broadcasts this COMMIT quorum certificate to other nodes in the DECIDE message. When nodes receive this DECIDE message, they execute the request because this message contains an already committed certificate/value. The new view starts once the state transition occurs due to the DECIDE message acceptance and execution.

We can visualize this protocol in Figure 8-5.



R1 (leader) broadcasts Msg(prepare, curProposal , high QC) R2,R3,R4 send voteMsg(prepare, m.node , ⊥) to R1

R1 (leader) broadcast Msg(pre-commit, ⊥, prepareQC) R2,R3,R4 (replicas) send voteMsg(pre-commit, m.justify .node,⊥) to R1

R1(leader) broadcast Msg(commit, ⊥, pre-commit QC) R2,R3,R4 (replicas) send voteMsg(commit, m.justify .node , ⊥) to R1

R1 (leader) broadcast Msg(decide, ⊥, commit QC) R2,R3,R4 (replicas) execute & respond to clients

When a new leader has collected new-view messages from (n - f) nodes, the protocol for new leader starts and the leader broadcasts prepare message.

Upon the leader receiving (n - f) prepare votes, it creates a quorum certificate (QC) called 'prepare quorum certificate'

Upon the leader receiving (n - f) pre-commit votes, it creates a pre-commit quorum certificate and broadcast it to other nodes as the commit message

Upon the leader receiving (n - f) commit votes. It creates a commit quorum certificate. This commit quorum certificate is broadcast to other nodes in the decide message

Figure 8-5. *HotStuff protocol*

More precisely, HotStuff steps are listed as follows:

- A new primary acquires new view messages from n-f nodes with the highest prepare quorum certificate that each validator receives. The primary looks at these messages and finds the prepare QC with the highest view (round number). The leader then broadcasts the proposal in a prepare message.
- When other nodes receive this prepare message from the leader, they check if the prepare proposal extends the highest prepare QC branch and has the higher view number associated than what they have currently locked.

- Other nodes reply to the leader with an acknowledgment.
- The leader collects the acknowledgments from $n - f$ prepare votes.
- When n-f votes are acquired by the leader, it combines them into a prepare QC and broadcasts this QC in a pre-commit message.
- Other replicas reply to the leader with a pre-commit vote. When the leader has received n-f pre-commit votes from other nodes, the primary combines them into a pre-commit QC and broadcasts them in a commit message.
- Replicas reply to the leader with commit votes and replicas lock on the pre-commit QC. When the leader receives n-f commit votes from the replicas, it combines them into a commit QC and broadcasts the decide message.
- When the nodes receive a decide message, they execute the operations/commands and start the next view.
- This operation repeats.

There are other optimizations such as pipelining which allows for further performance improvements. As all the phases are fundamentally identical, it's easy to pipeline HotStuff, which improves performance. Pipelining allows the protocol to commit a client's request in each phase. In a view, a leader in each phase proposes a new client request. This way, a leader can concurrently process pre-commit, commit, and decide messages for previous client requests passed on to the last leader via the commit certificate.

Safety and Liveness

HotStuff guarantees liveness by using the pacemaker, which ensures progress after GST within a bounded time interval by advancing views. This component encapsulates view synchronization logic to ensure liveness. It keeps enough honest nodes in the same view for sufficiently long periods to ensure progress. This property is achieved by progressively increasing the time until progress is made.

Whenever a node times out in a view, it broadcasts a timeout message and advances to the following when a quorum certificate of $2f + 1$ timeout messages is received. This certificate is also sent to the next leader, who takes the protocol further. Does this timeout detection sound familiar? It sounds familiar because pacemaker abstraction is basically a failure detector that we discussed before in Chapter 3. Moreover, voting and relevant commit rules ensure safety in HotStuff.

HotStuff is a simple yet powerful protocol that combines several innovations to produce a better protocol than its predecessors.

Polkadot

Polkadot is a modern blockchain protocol that connects a network of purpose-built blockchains and allows them to operate together. It is a heterogenous multichain ecosystem with shared consensus and shared state.

Polkadot has a central main chain, called a relay chain. This relay chain manages the Parachains – the heterogenous shards that are connected to the relay chain. A relay chain holds the states of all Parachains. All these Parachains can communicate with each other and share the security, which leads to a better and more robust ecosystem. As the Parachains are heterogenous, they can serve different purposes; a chain can be a specific chain for smart contracts, another for gaming, another could be for providing some public services, and so on and so forth. The relay chain is secured by a nominated proof of stake.

The validators on the relay chain produce blocks and communicate with Parachains and finalize blocks. On-chain governance decides what the ideal number of validators should be.

A depiction of the Polkadot chain with Parachains is shown in Figure 8-6.

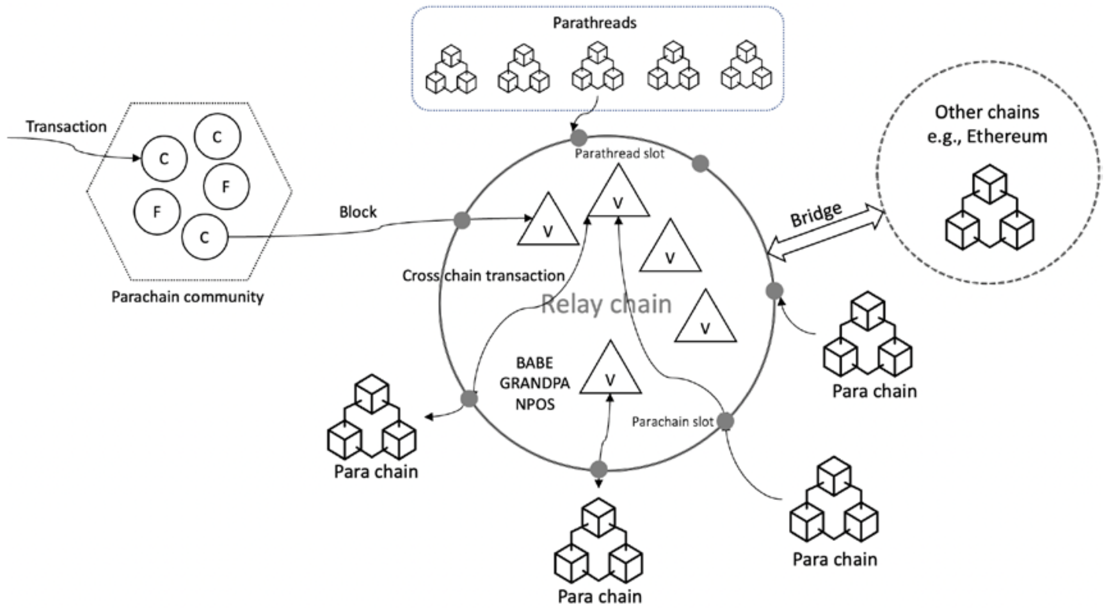


Figure 8-6. *A simple depiction of Polkadot*

Polkadot aims to be able to communicate with other blockchains as well. For its purposes, bridges are used, which connect Parachains to external blockchains, such as Bitcoin and Ethereum.

There are several components in Polkadot. The **relay chain** is the main chain responsible for managing Parachains, cross-chain interoperability, and interchain messaging, consensus, and security.

It consists of nodes and roles. Nodes can be light clients, full nodes, archive nodes, or sentry nodes. Light clients consist of only the runtime and state. Full nodes are pruned at configurable intervals. Archive nodes keep the entire history of blocks, and sentry nodes protect validators and thwart DDoS attacks to provide security to the relay chain. There are several roles that nodes can perform: validator, nominator, collator, and fisherman. **Validators** are the highest level in charge in the system. They are block producers, and to become block producers, they need to provide a sufficient bond deposit. They produce and finalize blocks and communicate with Parachains. **Nominators** are stakeholders and contribute to the validators' security bond. They place trust in a validator to "be good" and produce blocks. **Collators** are responsible for transaction execution. They create unsealed but valid blocks to validators that propose. **Fishermen** are used to detect malicious behavior. Fishermen are rewarded for providing proof of misbehavior of participants. **Parachains** are heterogenous blockchains connected to the relay chain. These are fundamentally the execution core of Polkadot. Parachains can be with their own runtime called application-specific blockchains. Another component called **Parathread** is a blockchain that works within the Polkadot host and connects to the relay chain. They can be thought of as pay-as-you-go chains. A Parathread can become a Parachain via an auction mechanism. **Bridges** are used to connect Parachains with external blockchain networks like Bitcoin and Ethereum.

Consensus in Polkadot

Consensus in Polkadot is achieved through a combination of various mechanisms. For governance and accounting, a nominated proof of stake is used. For block production, BABE is used. GRANDPA is the finality gadget. In the network, validators have their own clocks, and a partially synchronous network is assumed.

Finality is usually probabilistic as we saw in traditional Nakamoto PoW consensus. In most permissioned networks and some public networks, it tends to be deterministic, that is, provable finality, for example, PBFT and Tendermint.

In Polkadot, due to the reason that it is a multichain heterogenous architecture, there could be some situations where, due to some conflicts between chains, some rogue blocks are added. These rogue blocks will need to be removed after conflict resolution; in such situations, deterministic finality is not suitable due to its irreversible property. On the other hand, PoW is too slow, energy consuming, and probabilistic. The solution for this is to keep producing blocks as fast as possible but postpone finality for later as soon as it is suitable to finalize. This way, block production can continue and is revertible, but finality decision can be made separately and provably at a later stage.

This notion of provable finality is quite useful in a multichain heterogenous network because it allows us to prove to other parties that are not involved in consensus that a block is final. Also, provable finality makes it easier to make bridges to other blockchains.

This hybrid approach works by allowing validators to produce blocks even if only one validator is online and correct, but the finalization of the blocks is offloaded to a separate component called a finality gadget. Under normal conditions, block finalization is also quite fast, but in case of issues such as state conflicts, the finalization can be postponed until more scrutiny checks are performed on the blocks. In case of severe attacks or huge network partitions, block production will continue; however, as a fallback mechanism, Polkadot will fall back to the probabilistic finalization mechanism. This way, liveness is guaranteed even under extreme scenarios, as long as at least one validator is correct and alive. The blocks are produced by BABE, whereas they are finalized by GRANDPA. GRANDPA finalizes a chain of blocks instead of block-by-block finalization which improves efficiency. As finalization is a separate process, block production can continue at whatever speed the network allows for, but finality doesn't impact the block production speed and is done later.

There can be some forks before a "best" chain is finalized by GRANDPA. We can visualize this in Figure 8-7.

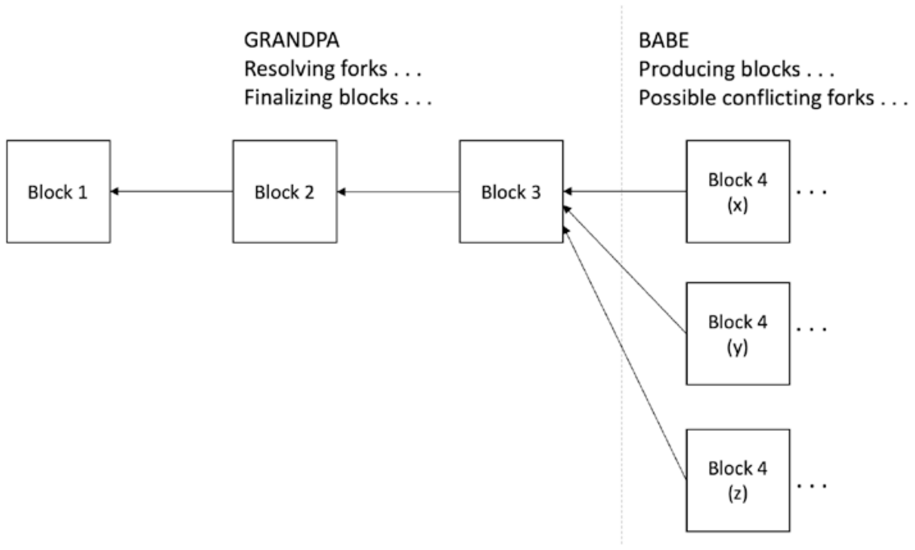


Figure 8-7. *BABE producing and GRANDPA finalizing*

The diagram shows on the right side three produced blocks by BABE in three forks; GRANDPA resolves these forks and finalizes the chain. Now let's see how blocks are produced by BABE.

BABE – Blind Assignment for Blockchain Extension

BABE is a proof of stake protocol for block production where validators are randomly selected based on their staked amount to produce blocks. It does not depend on any central clock.

Time is divided into periods spanning n seconds called slots, where a block is expected to be produced. An epoch is a sequence of slots. At each slot, validators run a VRF, which, based on randomness generated from previous epochs, decides whether to produce a block or not. We can visualize this in Figure 8-8.

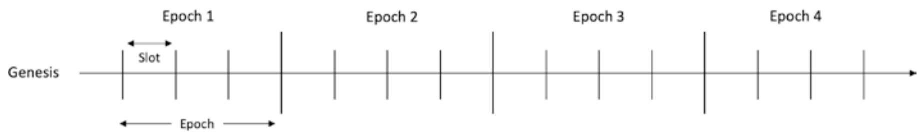


Figure 8-8. *Slots and epochs*

Validators use public key cryptography. There are two types of key pairs. A private key from the first key pair is used for block signing. The second pair is used for a verifiable random function (VRF), also called the lottery key pair. A private key from the latter pair is used as an input to the verifiable random function. Block signing provides usual nonrepudiation, integrity, and data origin authentication guarantees, verifying that the validator has indeed produced this block. In the VRF, the private key generates the randomness, but the public key proves to other nodes that the randomness generated is indeed reliable and that the validator did not cheat.

Each validator has an almost equal chance of being selected. A slot leader election is done like PoW, where, if the result of VRF is lower than a predetermined threshold, then the validator wins the right to produce the block. Also, the proof generated from the VRF enables other participants to verify that the validator is following the rules and not cheating; in other words, it proves that the randomness generated is reliable. If the value produced from the VRF is higher than or equal to the target, then the validator simply collects blocks from other validators.

Here are the phases of the BABE protocol.

Genesis Phase

In this phase, the unique genesis block is created manually. A genesis block contains a random number that is used during the first two epochs for the slot leader selection.

Normal Phase

Each validator divides its time into so-called slots after receiving the genesis block. Validators determine the current slot number according to the relative time algorithm, which we'll explain shortly. Each validator during normal operation is expected to produce a block, whereas other nonvalidator nodes simply receive the produced blocks and synchronize. It is expected that each validator has a set of chains in the current slot/epoch and has the best chain selected in the previous slot by using the best chain selection mechanism, which we'll explain shortly.

The slot leader selection is based on the output of the VRF. If the output of the VRF is below a certain threshold, then the validator becomes the slot leader. If not, then it simply collects the blocks from the leader.

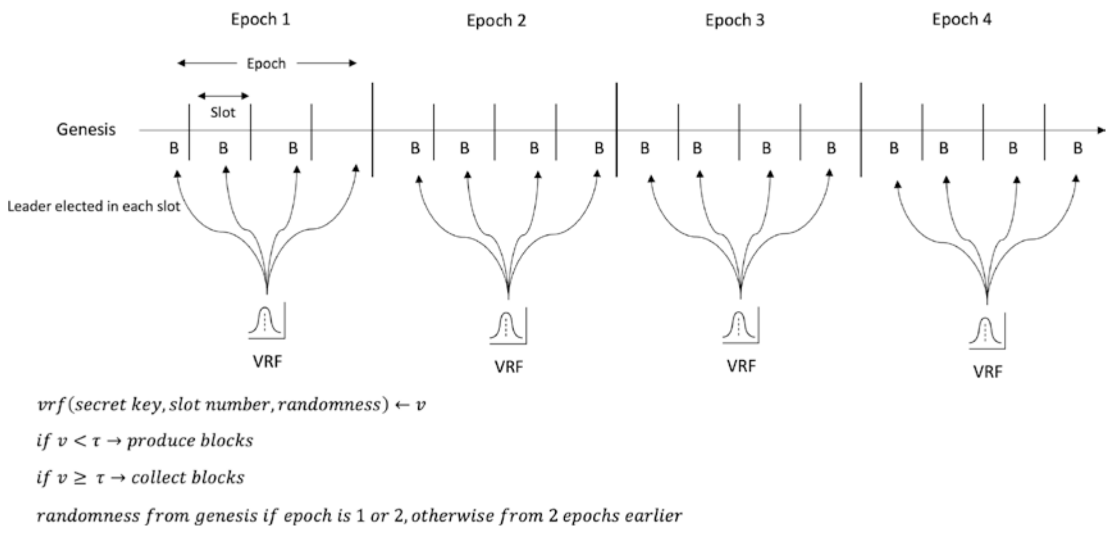
The block generated by the leader is added to the best chain selected in the current slot. The produced block must at least contain the slot number, the hash of the previous block, the VRF output, VRF proof, transactions, and the digital signature. Once the chain is updated with the new block, the block is broadcast. When another non-leader validator receives the block, it checks if the signature is valid. It also verifies if a valid leader has produced the block by checking the VRF output using the VRF verification algorithm. It checks that if the output of the VRF is lower than the threshold, then the leader is valid. It further checks if there is a valid chain with the required header available in which this received block is expected to be added, and if the transactions in the block are valid.

If all is valid, then the validator adds the block to the chain. When the slot ends, the validator finally selects the best chain using the best chain selection algorithm, which eliminates all chains that do not include the finalized block by the finality gadget GRANDPA.

Epoch Update

A new epoch starts every n number of slots. A validator must obtain the new epoch randomness and active validator set for the new epoch before beginning the new epoch. The new validator set for the new epoch is included in the relay chain to enable block production. A new validator must wait for two epochs before the protocol can select it. Adding a validator two epochs later ensures that VRF keys of the new validators are added to the chain before the randomness of the future epoch in which they are going to be active is revealed. A new randomness for the epoch is calculated based on the previous two epochs by concatenating all the VRF outputs of blocks in those epochs.

The diagram in Figure 8-9 illustrates this slot leader election process.



$vrf(secret\ key, slot\ number, randomness) \leftarrow v$

$if\ v < \tau \rightarrow produce\ blocks$

$if\ v \geq \tau \rightarrow collect\ blocks$

$randomness\ from\ genesis\ if\ epoch\ is\ 1\ or\ 2, otherwise\ from\ 2\ epochs\ earlier$

***Figure 8-9.** Slot leader election via VRF and block production in slots and epochs*

The best chain selection algorithm simply removes all chains that do not contain a finalized block by GRANDPA. In case GRANDPA does not finalize any block, the protocol falls back to probabilistic finality, and the finalized block is chosen as the one which is several blocks (a number) before the last block. This works almost like a chain depth rule in PoW.

Time is managed in BABE using the **relative time algorithm**. It is critical for the security of the BABE that all parties are aware of the current slot number. BABE does not use a time source managed by NTP as clearly a central source of time cannot be trusted. Validators realize a notion of logical time by using block arrival times as a reference without relying on an external source of time. When a validator receives the genesis block, it records the arrival time as a reference point of the beginning of the first slot. As the beginning time of each slot is expected to be different on each node, an assumption is made that this difference is reasonably limited. Each validator updates its clock by calculating the median of the arrival times of the blocks in the epoch. Although the mechanics are different, the fundamental concept appears to be similar to the logical clocks we discussed in Chapter 1. Temporary clock adjustment until the next epoch is also possible for validators that went offline and joined the network again.

Safety and Liveness

There are four security properties that BABE satisfies: chain growth, existential chain quality, chain density, and common prefix.

Chain Growth

This property guarantees a minimum growth between slots. In other words, it's a liveness property, and chain growth is guaranteed as long as a supermajority of honest validators is available. Malicious validators cannot stop the progress of the best chain.

Chain Quality

This property ensures that at least one honest block is contributed to any best chain owned by an honest party in every x number of slots. The protocol guarantees that even in the worst case, there will be at least one honest block included in the best chain during an epoch. This ensures that the randomness is not biased.

Chain Density

This property ensures that any sufficiently long portion of blocks in the best chain contains more than half of the blocks produced by honest validators. This property is implied by chain growth and chain quality.

Common Prefix

This property ensures that any blocks before the last block in the best chain of an honest validator cannot be changed and are final. Again, this property is satisfied due to the assumption of super honest majority of honest validators. It is rare for a malicious validator to be elected in a slot, and only mostly honest validators will be elected; therefore, malicious validators are in such a minority that they cannot create another "best" chain which do not contain a finalized block.

GRANDPA – GHOST-Based Recursive Ancestor Deriving Prefix Agreement

It is a gadget that finalizes the blocks separately after they have been produced by BABE. It is essentially a Byzantine agreement protocol that agrees on a chain out of many forks. The difference here is that usually in BFT protocols, a decision is made on a single block, whereas GRANDPA decides on a chain of blocks (a fork) and decides what the final chain is. It's a finalization mechanism that resolves forks.

GRANDPA assumes a partially synchronous network. It works in rounds with each round having 3f+1 eligible voters out of which 2f+1 voters are assumed honest. In other words, it requires that two-thirds of the validator set is honest and agrees on a prefix of the canonical chain, which ultimately becomes finalized. In each round, a primary is pseudorandomly elected, which is agreed upon by the participants. Moreover, all participants agree on the voter set. Primary selection can also be based on rotation between voters.

The protocol works in two phases: pre-vote and pre-commit. The pre-vote allows validators to estimate what can be finalized, that is, validators pre-vote on a best chain. For the pre-commit, validators use the two-thirds GHOST rule to pre-votes collected and pre-commit. Finally, the pre-commits are finalized.

GRANDPA consists of two protocols. The first protocol works under partial synchrony and tolerates one-third Byzantine faults. The second protocol works in a fully asynchronous environment and can tolerate one-fifth Byzantine faults.

GRANDPA Protocol Steps

In each round, a participant is selected as a leader (primary), and other participants are also aware who the primary is:

- The new round starts.
- The primary broadcasts the highest block that it thinks might be final from the previous round.
- Validators wait for a certain network delay, then each validator broadcasts a “pre-vote” message for the highest block that it believes should be finalized. If validators receive a block from the primary with the better best chain, then the validators use that best chain. If the supermajority of the validators is correct, this block is expected to extend the chain that the primary has broadcast.
- Each validator looking at the pre-votes determines the highest block that could be finalized. If the pre-votes extend the last finalized chain, each validator will cast a pre-commit to that chain.
- Each validator waits for sufficient pre-commits to compose a commit message on the newly finalized chain.

The protocol penalizes misbehavior by slashing where a percentage of the stake is deducted for malicious behavior or even just irresponsible behavior, for example, a long-running inactive node. An increase in penalties is proportional to the number of participants.

Safety

The protocol ensures that all votes are descendants of some block that could have been finalized in the previous round. Nodes estimate the finalization possibility of a block based on pre-votes and pre-commits. Before a new round starts, nodes ensure by acquiring enough pre-commits that no block with this round's estimate can be finalized on a different chain or later on the same chain. In the next round, it also ensures that it only pre-votes and pre-commits on the blocks that are descended from the last round's estimate.

Liveness

The protocols select a validator in rotation to become the primary. The primary starts the round by broadcasting their estimate of block finalization from the last round. Validators pre-vote for the best chain, including the primary's proposed block if (1) the block is at least the validator's estimate and (2) the validator has acquired >2/3 pre-votes for the block and its descendants in the last round.

The key insight here is that if the primary's proposed block has not been finalized, it is finalized to make progress. For example, suppose the proposed block by the primary has not been finalized, and all validators have agreed on the best chain with the last finalized block. In that case, progress is made by finalizing the latest agreed final chain. If GRANDPA cannot conclude, then BABE provides its probabilistic finality as a fallback mechanism, ensuring progress.

GRANDPA and BABE are one of the latest heterogenous multichain protocols. There are other protocols in this family such as Casper FFG used in the Ethereum consensus layer (Ethereum 2, beacon chain).

Ethereum 2

Ethereum 2, also called Serenity or Eth2, is the final version of Ethereum. Currently, Ethereum is based on proof of work and is known as Eth1. It is now called the execution layer, and the previous terminology of Eth1 and Eth2 is no longer valid. Eth2 is now called the consensus layer. As per the original plan, the existing PoW chain will eventually be deprecated, and users and apps will migrate to the new PoS chain Eth2. However, this process is expected to take years, and an alternative better proposal is to continue improving the existing PoW chain and make it a shard of Ethereum 2. This change will ease the transition to proof of stake and allow scaling up using rollups instead of sharded execution. The beacon chain is already available; "the merge" phase where the Ethereum mainnet merges with the beacon chain is expected in 2022. After the merge, the beacon chain will become executable with proof of stake and EVM capabilities. Old Ethereum 1 (Eth1) will become the execution layer with execution clients, for example, "geth" from Eth1. Ethereum 2 (consensus) clients such as prysm and lighthouse will continue operating on the beacon chain. Eventually, the shard chains that expand Ethereum capacity and support execution are planned for 2023. Ethereum 2.0 "consensus" with Ethereum 1 "execution" as shard 0, along with other upgrades based on their road map, can be visualized in Figure 8-10.

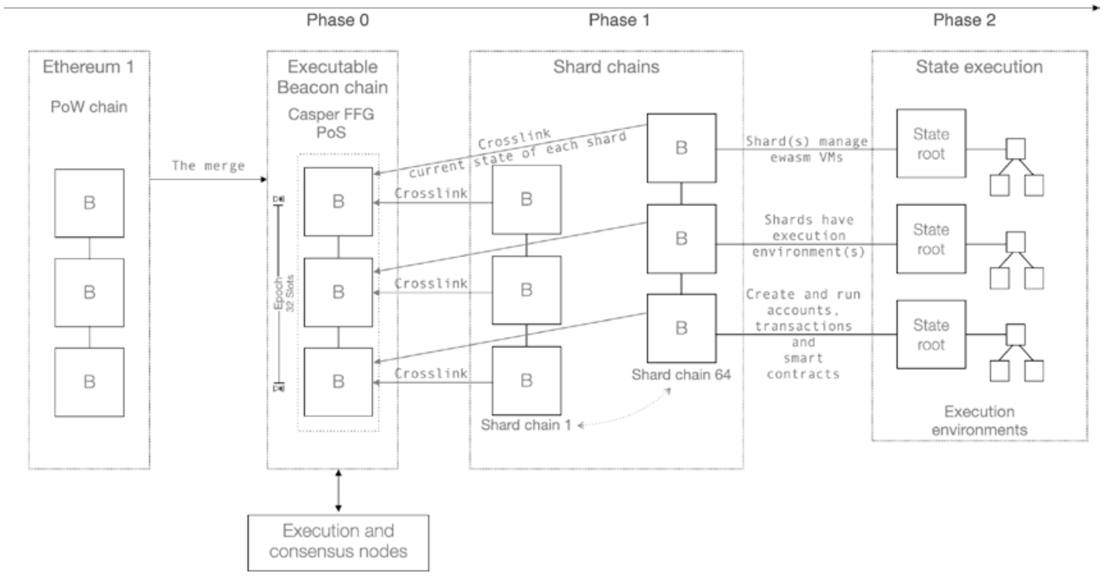


Figure 8-10. *Ethereum upgrades, showing the merge and subsequent upgrades*

In short, Eth1 is now called the execution layer, which handles transactions and executions, whereas Eth2 is now called the consensus layer, which manages proof of stake consensus. As part of the consensus layer, the "Ethereum 2" proof of stake consensus protocol is proposed, which we discuss next.

Casper

Casper is a proof of stake protocol which is built to replace the current PoW algorithm in Ethereum. There are two protocols in this family:

- Casper the Friendly Finality Gadget (FFG)
- Casper the Friendly GHOST

Casper FFG is a PoS BFT–style hybrid protocol that adds a PoS overlay to the current PoW, whereas Casper the Friendly GHOST is purely a PoS protocol. Casper FFG provides a transition phase before being replaced with Casper CBC, a pure PoS protocol. We'll discuss Casper FFG as follows.

Casper FFG

Casper can be seen as an improved PBFT with proof of stake for public blockchains. Casper the Friendly Finality Gadget introduces some novel features:

- Accountability
- Dynamic validators
- Defenses
- Modular overlay

Casper FFG is an overlay mechanism on top of a block proposing mechanism. Its sole purpose is consensus, not block production.

Accountability allows to detect rule violations and identify the validators who violated the rule. This enables the protocol to penalize the violating validator, which serves as a defense against malicious behavior.

Casper FFG also introduces a safe way to change (i.e., add/remove) participants' validator set.

The protocol also introduces a defense mechanism to protect against network partitions and long-range attacks. Even if more than one-third of validators go offline, the protocol provides a defense against such scenarios.

Casper FFG is an overlay, which makes it easier to add on top of an existing PoW chain. As it is an overlay, it is expected that the underlying chain has its own fork choice rule. In the Ethereum beacon chain, it is an overlay on top of the fork choice rule called LMD GHOST (Latest Message Driven Greedy Heaviest-Observed Sub-Tree).

The LMD GHOST fork choice rule is based on GHOST. The LMD GHOST choice rule selects the correct chain from multiple forks. The honest chain is the one that has the most attestations from the validators and stake (i.e., weight). Forks occur due to network partitions, Byzantine behavior, and other faults. We can see LMD GHOST in action in Figure 8-11.

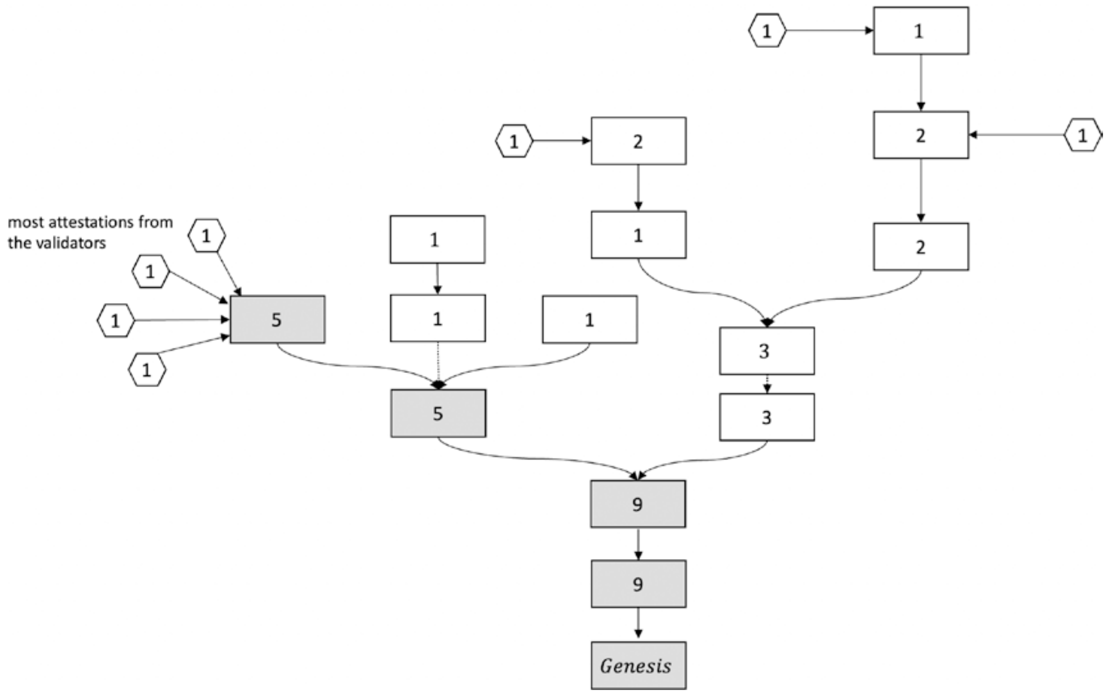


Figure 8-11. *LMD GHOST fork choice rule*

In Figure 8-11

- The number in the block represents the weight by stake in blocks.
- The hexagon represents attestations from validators carrying weight 1.
- The canonical chain is the one with shaded blocks (weight + attestations).

Usually, when proof of work or some other blockchain production mechanism produces blocks, they are produced one after another, in a sequential and linear chain of blocks, where each parent has exactly one child. But it can happen due to network latency and faulty/Byzantine nodes that the proposal mechanism produces multiple child blocks of a parent block. Casper is responsible for choosing a single child from each parent block, thus choosing a single canonical chain from the block tree.

Casper however does not deal with the entire block tree due to efficiency concerns; instead, it considers a subtree of checkpoints forming the checkpoint tree.

New blocks are appended to the block tree structure. A subtree of a tree called the checkpoint tree is where the decision is required. This structure is shown in Figure 8-12.

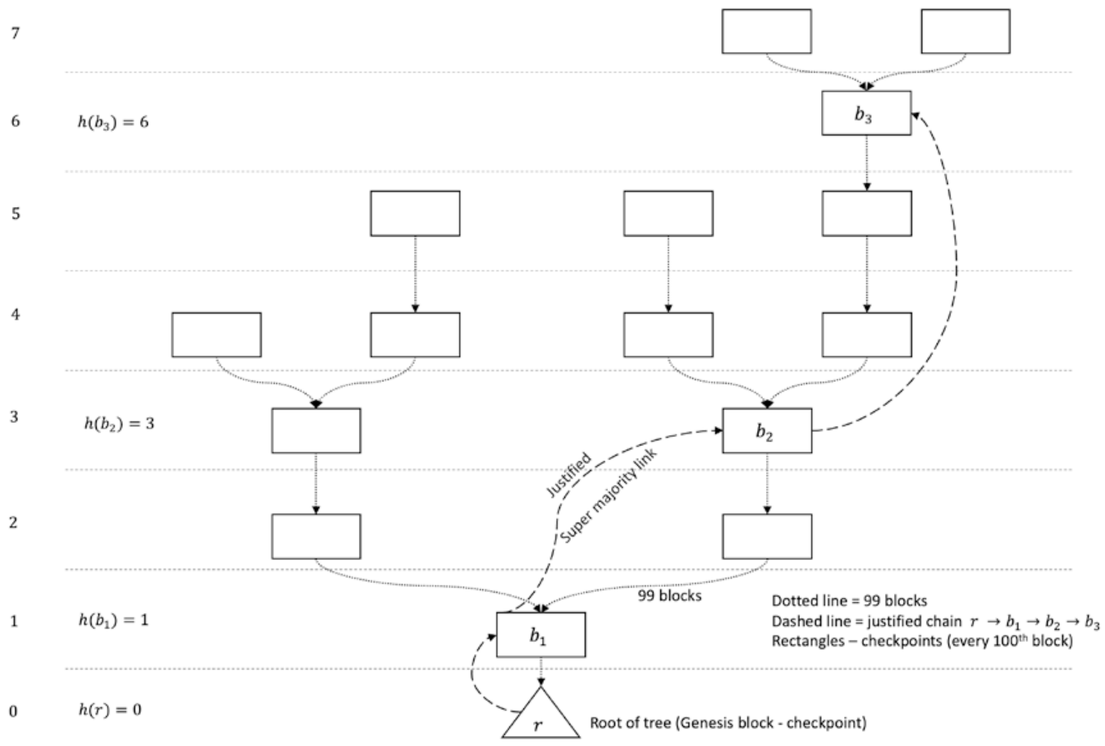

7
6
$h(b_3) = 6$
b_3
5
4
3
$h(b_2) = 3$
b_2
Justified
Super majority link
2
1
$h(b_1) = 1$
b_1
99 blocks
Dotted line = 99 blocks
Dashed line = justified chain $r \rightarrow b_1 \rightarrow b_2 \rightarrow b_3$
Rectangles – checkpoints (every 100th block)
0
$h(r) = 0$
r
Root of tree (Genesis block - checkpoint)

Figure 8-12. *Checkpoint tree showing heights, votes, checkpoints, and canonical chain*

The genesis block is a checkpoint, and every 100th block is a checkpoint. The distance from one checkpoint to another is called an epoch. In other words, validators finalize checkpoints every 100 blocks. Each validator that joins the network deposits their owned deposit. This deposit is subject to increase and decrease due to penalty and reward mechanism. Validators broadcast votes. A vote weight is proportional to a validator's stake. A validator can lose the entire deposit if it deviates from the protocol, that is, violates any rules. This is to achieve safety.

A vote message has some attributes, which we describe as follows:

- **s**: Hash of a justified source checkpoint
- **t**: Hash of the target checkpoint to be justified
- **h(s)**: Height of the source checkpoint
- **h(t)**: Height of the target checkpoint
- **S**: Signature on the complete vote message from the sender

Validators create the vote message, sign it, and broadcast to the network.

The hash of a checkpoint is used to identify the corresponding checkpoint. The vote is valid only if s is an ancestor of t in the checkpoint tree and the public key of the validator is in the validator set. When more than two-thirds of validators vote on a chain from a source to a target, this chain or link becomes the supermajority link. For example, cp' as a source and cp as a target, then $cp' \rightarrow cp$ is the majority link. A checkpoint is justified if it is the genesis block or if it is the checkpoint in a supermajority link where the last checkpoint is justified. Precisely, we can say a checkpoint cp is justified if there is a supermajority link $cp' \rightarrow cp$ where cp' is justified. A checkpoint cp is considered final if it is justified and there exists a supermajority link $cp \rightarrow cp'$ where cp' is a direct child of checkpoint cp.

Justified checkpoints are not considered final as there can exist conflicting justified checkpoints. To finalize a checkpoint cp, a second round of confirmation is required where a direct child cp' of cp with a supermajority link $cp \rightarrow cp'$ is justified.

Protocol steps

- Token holders deposit a stake in the network.
- During a consensus epoch
  - Identify valid checkpoints.
  - Add them to the checkpoint tree.
  - Broadcast a vote for a checkpoint pair (source, target).
  - Evaluate received votes' validity, check against slashing rules and the deposited stake of the vote signer.

    - If the checkpoint pair has more than two-thirds of total deposited stakes, then the checkpoint is considered validated.
    - The checkpoint pair is justified and finalized.
- Repeat the process.

Essentially, the process works in three steps. First, votes are casted for a checkpoint; if >2/3 votes are acquired, then the checkpoint is in a justified state. Finally, the chain is finalized to form the canonical chain. Justification does not mean finalized; there can be multiple justified chains in the block tree. Finality occurs when two consecutive checkpoints receive 2/3 votes.

Safety and liveness

Any validator who violates these conditions will be penalized by slashing their deposit. These are known as minimum slashing conditions:

- A validator must not publish two distinct votes for the same target checkpoint height.
- A validator must not vote again within the source to target span of its other existing vote.

There are two safety and liveness requirements:

- Accountable safety
- Plausible liveness

Accountable safety is a little different from the traditional safety requirement in consensus protocols. Safety simply means that two conflicting checkpoints can never be finalized unless more than one-third of validators deviate from the protocol. Accountability means that misbehaving validators can be identified and penalized.

Plausible liveness means that as long as there exist children that extend the finalized chain, supermajority links can always be added to produce new finalized checkpoints.

Summary

This chapter discussed the blockchain age protocols that emerged after Bitcoin. There are several types of blockchain consensus protocols; some are based on voting, some are proof of work, and another class is proof of stake protocols. All these protocols have some safety and liveness properties that ensure that the protocol is correct and works correctly in a given environment. We discussed several of these points in this chapter. A concluding comparison of these protocols is presented in the last chapter, Chapter 10.

We also discussed protocols such as ETHASH, proof of stake, its different types, and BFT variations, including HotStuff and Tendermint, in detail. Modern protocols such as Casper FFG, Solana's proof of history, and Polkadot's GRANDPA and BABE were also introduced. It is impossible to cover all protocols in this chapter, but at least adequate information is given to build a good understanding of different types of algorithms used in the blockchain. Some more protocols that we did not cover, such as PoET, HoneyBadger BFT, and Snow, will be briefly presented in the last chapter.

This area is a very active field of research, and many researchers in academia and industry are very much interested in this area. As such, only further evolution and advancement is expected in this space.

In the next chapter, we will discuss another exciting topic, quantum consensus, a subject that has emerged recently with the advent of quantum computing.

Bibliography

1. Peer coin paper: www.peercoin.net/whitepapers/peercoin-paper.pdf
2. Xiao, Y., Zhang, N., Lou, W., and Hou, Y.T., 2020. A survey of distributed consensus protocols for blockchain networks. IEEE Communications Surveys & Tutorials, 22(2), pp. 1432–1465.
3. Bashir, I., 2020. Mastering Blockchain: A deep dive into distributed ledgers, consensus protocols, smart contracts, DApps, cryptocurrencies, Ethereum, and more. Packt Publishing Ltd.
4. "Hidden lock problem": http://muratbuffalo.blogspot.com/2019/12/hotstuff-bft-consensus-in-lens-of.html
5. Web3 Foundation Consensus Tutorial – Bill Laboon: https://youtu.be/1CuTSluL7v4
6. https://polkadot.network/blog/polkadot-consensus-part-2-grandpa/

7. Buterin, V. and Griffith, V., 2017. Casper the friendly finality gadget. arXiv preprint arXiv:1710.09437.

8. "Replace communication with local computation" – Liskov, B., 1993. Practical uses of synchronized clocks in distributed systems. Distributed Computing, 6(4), pp. 211–219.

9. Buterin, V., Hernandez, D., Kamphefner, T., Pham, K., Qiao, Z., Ryan, D., Sin, J., Wang, Y., and Zhang, Y.X., 2020. Combining GHOST and casper. arXiv preprint arXiv:2003.03052.

10. Solana whitepaper: Solana: A new architecture for a high performance blockchain v0.8.13: https://solana.com/solana-whitepaper.pdf

11. Burdges, J., Cevallos, A., Czaban, P., Habermeier, R., Hosseini, S., Lama, F., Alper, H.K., Luo, X., Shirazi, F., Stewart, A., and Wood, G., 2020. Overview of polkadot and its design considerations. arXiv preprint arXiv:2005.13456.

12. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., and Abraham, I., 2018. HotStuff: BFT consensus in the lens of blockchain. arXiv preprint arXiv:1803.05069.

13. Buterin, V., Hernandez, D., Kamphefner, T., Pham, K., Qiao, Z., Ryan, D., Sin, J., Wang, Y., and Zhang, Y.X., 2020. Combining GHOST and casper. arXiv preprint arXiv:2003.03052.

14. Stewart, A. and Kokoris-Kogia, E., 2020. GRANDPA: a Byzantine finality gadget. arXiv preprint arXiv:2007.01560.

15. Buchman, E., 2016. Tendermint: Byzantine fault tolerance in the age of blockchains (Doctoral dissertation, University of Guelph).