Consensus Algorithms for Blockchain

Hyunsoo Kim and Taekyoung Ted Kwon

Abstract A consensus algorithm is an essential component of a blockchain, responsible for reaching an agreement among decentralized nodes. It also determines the performance and characteristics of an application. With more than 2,000 different cryptocurrencies currently in use, we face an ever-growing list of consensus algorithms. Furthermore, the inherent complexity of consensus algorithms and their rapid evolutions make it hard to assess their suitability for blockchain applications. Understanding the pros and cons of a consensus algorithm is crucial in designing new blockchain services and developing more advanced algorithms. We propose a framework with comprehensive criteria to evaluate consensus algorithms in terms of performance, security, and decentralization. In addition, we present the operational mechanisms and analyze the characteristics of mainstream consensus algorithms, namely, proof-based algorithms such as Proof of Work (PoW) and Proof of Stake (PoS), and vote-based algorithms with Byzantine Fault Tolerance (BFT). The algorithms are evaluated based on our proposed framework to provide a better understanding. We hope this article leads us to identify research challenges and opportunities of consensus algorithms.

1 Introduction

Blockchain technologies have received widespread attention across the industry, governments, and academia alike over the past decade. Today's most predominant blockchain applications are cryptocurrencies, for instance, Bitcoin has recently hit $1 trillion in market value [1]. Cryptocurrencies have disrupted the long-established centralized financial system on a global scale. Many developing countries are now

H. Kim · T. T. Kwon (✉)
Department of Computer Science and Engineering, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul, South Korea
e-mail: tkkwon@snu.ac.kr

H. Kim
e-mail: wayles@snu.ac.kr

D. A. Tran et al. (eds.), *Handbook on Blockchain*, Springer Optimization and Its Applications 194, https://doi.org/10.1007/978-3-031-07535-3_3

seeing higher rates of cryptocurrency adoption. Take Nigeria, for example; roughly a third of the population owns cryptocurrencies and uses them in everyday lives [2].

Now, we are witnessing the blockchain expands across various industries such as energy, health care, real estate, supply chain, and so on. According to a recently conducted study [3], blockchain technologies have the potential to boost global Gross Domestic Product (GDP) by $1.76 trillion across the industry over the next decade, which is 1.4% of the predicted global GDP.

Blockchain is essentially a decentralized, asynchronous distributed system, often with much more nodes than its traditional counterpart. Making reliable communications between the nodes and maintaining the correct state across the system even in the presence of malicious nodes and network failures are the key issues [4]. This is where a consensus algorithm takes place. At the heart of a blockchain (or its application), the consensus algorithm is responsible for maintaining consistent copies of the current state across all nodes, validating new transactions, and updating the current state while achieving an agreement among the nodes.

The Proof-of-Work (PoW) consensus algorithm used in Bitcoin is the first and most popular consensus algorithm in the blockchain. However, although this consensus algorithm is well-fitted for the application of Bitcoin, it has its shortcomings. Namely, its energy inefficiency of validating and constructing a new block, known as *mining*, and low throughput have been a vexing issue [5]. As a result, researchers and developers have sought to devise new consensus algorithms.

There are over 2,000 different cryptocurrencies, let alone blockchain applications from other industries, which are currently employing diverse consensus algorithms. The list of consensus algorithms is extensive, and even now, newer ones are under way. Researchers and developers must understand the characteristics and limitations of a consensus algorithm since the overlaying blockchain application's performance and usability will highly depend on it. Thus, we believe it is essential to lay out a framework that can be used to analyze and evaluate a consensus algorithm and determine its suitability to a particular application. The contributions of this article are summarized as follows. First, a framework for evaluating consensus algorithms is proposed. The framework consists of comprehensive criteria that can be applied to most consensus algorithms. Second, an in-depth survey of representative consensus algorithms and their characteristics are discussed.

The rest of this article is organized as follows. In Sect. 2, we first review the literature on consensus algorithms with a focus on their evaluation criteria. And then, we present our evaluation framework and discuss each criterion in detail. Section 3 presents major consensus algorithms categorized in PoW, PoS, and vote-based. A thorough evaluation will follow in Sect. 4 based on our proposed framework. Section 5 proposes future research opportunities regarding consensus algorithms and concludes the article.

2 Evaluation Criteria

With the advancement of different blockchain technologies and their applications in multiple domains, a variety of consensus algorithms have been developed. As most of the consensus algorithms have their limitations, there are still ongoing debates on addressing the drawbacks or issues of those consensus algorithms.

This paper aims to identify and provide key criteria for a consensus algorithm from diverse perspectives. In this section, we first review the major characteristics and functionalities of the consensus algorithms in the literature and then present a framework for their evaluation.

2.1 Related Works

In order to define and present an evaluation criteria framework, we first go over a comprehensive review of the prior consensus algorithms and their evaluations.

In [6], the authors focus on the Bitcoin cryptocurrency and list the following criteria: maximum throughput, latency, bootstrap time, cost per confirmed transaction, transaction validation, bandwidth, and storage. Their criteria are mostly related to the performance of the consensus algorithm with a focus on the Proof-of-Work (PoW) algorithm in Bitcoin.

Mingxiao et al. [7] compared the five consensus algorithms: PoW, Proof of Stake (PoS), Delegated Proof of Stake (DPoS), Practical Byzantine Fault Tolerance (PBFT), and Raft. The list of criteria consists of: Byzantine fault tolerance, crash fault tolerance, verification speed, throughput, and scalability. Note that the criteria can be classified into two categories: fault tolerance and performance.

Nguyen and Kim [8] performed a comprehensive survey of consensus algorithms by categorizing them to proof-based consensus algorithms: PoW, PoS, hybrid form of PoW and PoS, and voting-based consensus algorithms: Byzantine fault tolerance and crash fault tolerance. Performance comparison was done between PoW, PoS, and hybrid form of PoW and PoS based on energy efficiency, modern hardware, forking, double-spending attack, block creating speed, and pool mining. Another performance comparison was performed between proof-based consensus algorithms and vote-based consensus algorithms in general. The criteria were agreement making, joining nodes, number of nodes, decentralization, trust, node identity management, security threat, and reward. Although we take the similar categorization of consensus algorithms in this study, the listed criteria are focused on qualitative properties and only applied to each category and not individual algorithms.

Unlike the above studies, [9] did not categorize consensus algorithms for evaluation but viewed the blockchains at multiple levels: consensus level, mining pool, network level, and smart contracts. The authors specifically focused on attack vectors at the consensus level: double spending, Finney attack, Vector76 attack, brute force attack, 51% attack, and nothing-at-stake attack.

Table 1 Comparison of consensus algorithm evaluation criteria of [11, 12]

Ferdous et al. [11]		Bamakan et al. [12]	
Category	Criterion	Category	Criterion
Structural	Node type	Throughput	TPS
	Structure type		Block creation
	Underlying mechanism		Verification time
Block and reward	Genesis date		Block size
	Block reward	Profitability	Mining reward
	Total supply		Power consumption
	Block time		Transaction fees
Security	Authentication		Hardware dependency
	Non-repudiation	Decentralization	Blockchain governance
	Censorship resistance		Permission model
	Adversary tolerance		Trust model
	Sybil protection	Security	Double spending
	DoS resistance		51% attack
Performance	Fault tolerance		Sybil attack
	Throughput		
	Scalability		
	Latency		
	Energy consumption		

Bano et al. [10] performed a survey based on individual applications such as ByzCoin, Ouroboros, Bitcoin, and Spectre which is different from our approach. However, the authors classify the criteria into three categories: committee configuration, safety, and performance. Safety consists of censorship resistance, DoS resistance, and adversary tolerance, while performance consists of throughput (TPS), scalability, and latency.

Ferdous et al. [11] performed an extensive survey of consensus algorithms and classified them into two categories: incentivized consensus and non-incentivized consensus. This is analogous to our proof-based and vote-based categorization. [11, 12] are both noticeable for their structuring evaluation criteria of consensus algorithms, which is found in Table 1.

2.2 Evaluation Framework

Identifying universal criteria that apply to most, if not all, consensus algorithm is key to defining a solid evaluation framework. We also focus on generalized consensus

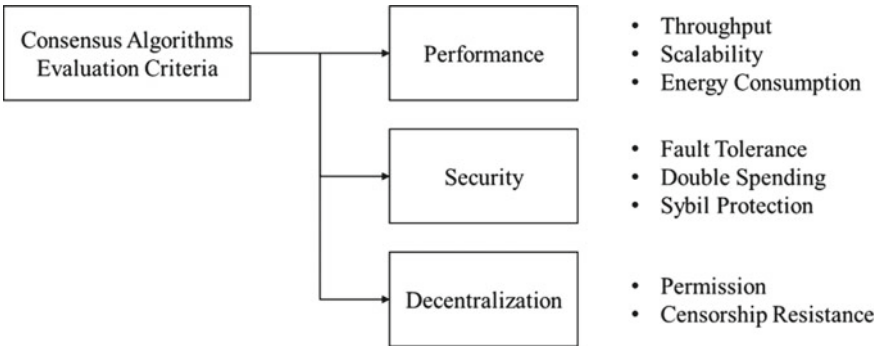

Fig. 1 The criteria of our evaluation framework for consensus algorithms are classified into three categories

algorithms such as PoW and PoS algorithms, not on individual cryptocurrency applications. As a result, structural criteria such as node characteristics and management, or profitability criteria, such as mining reward, mining pools, and transaction fees, are avoided unless necessary.

Based on the above standpoint and our review of the literature that define various criteria for consensus algorithm's performance evaluation, we present a framework to evaluate consensus algorithms in terms of criteria in the three following categories: performance, security, and decentralization, as depicted in Fig. 1. In the following, we will briefly introduce three categories and detail the criteria therein.

Performance Criteria

The performance criteria consist of properties or metrics to measure the quantitative performance of consensus algorithms. In this paper, we consider throughput, scalability, and energy consumption for performance.

Throughput/TPS

The *throughput* of a consensus algorithm is the speed of processing transactions by the participating nodes or members. In other words, the maximum throughput of a blockchain is the maximum rate at which the blockchain can confirm transactions [6]. It is also referred to as *Transactions per Second* (TPS), defined by the number of transactions processed per second. For example, if a particular blockchain processes an average of 600 transactions per minute, the TPS of that blockchain is 10 (10 transactions per second). The higher is the TPS, the faster the transactions will be verified, executed, and confirmed by that blockchain.

Throughput is one of the most essential criteria when discussing the performance of a blockchain. There are several elements that we should also consider when discussing throughput: *latency* and *block size.*

Latency: The *latency* refers to the time it takes from when a transaction is created to when the consensus (for the transaction) has been reached. In between, the transaction will be validated, added to the block, and appended to the chain. *Latency* is sometimes replaced by similar terms such as *block time* or *finality*, in which they have a slightly different meaning.

Block time is the time it takes to make a new block since the last block that was added to the chain. An increase in block time will increase the latency, effectively reducing the throughput.

There are two approaches in defining *finality*. One is *deterministic* (or *absolute*) *finality*, which guarantees that the transaction is verified and immutable as soon as it is added to the chain. The time to deterministic finality is identical to block time and, in most cases, latency as well [13]. Another, *probabilistic finality* is used when a transaction becomes probabilistically immutable as more blocks are added to the chain. This will be explored further in Double-Spending Prevention in Security Criteria (Section "Security Criteria").

Block size: The *block size* refers to the maximum amount of transactions (or bytes) in a block. Larger block size may lead to shorter latency since it can fill the block with more transactions. However, on the contrary, increasing the block size could improve the throughput of the consensus algorithm since more transactions can be included in the block given the same block time.

All in all, the throughput of a consensus algorithm is not determined by a single factor and must consider different variables and their implications. Table 2 presents the TPS, maximum block size, and the minimum and maximum latency of selected cryptocurrencies between March 2018 and February 2021 [14, 15].

Scalability

Scalability refers to the ability to support a growing number of users and nodes. It is considered one of the critical factors in the design of decentralized distributed systems. Throughput is also another aspect of scalability. As the network grows, we can expect the number of transactions to increase proportionally. However, the throughput limitation compared to centralized systems is one of the hindrances to

Table 2 TPS, latency, and max block size of selected cryptocurrencies [14, 15]

Cryptocurrency		Bitcoin	Ethereum	Litecoin	Ripple	Dogecoin	DASH	Monero
TPS		7	15	28	1500	16	56	30
Latency [min]	Min	7.35	0.22	2.12	0	1,03	2.56	1.57
	Max	15.65	0.39	3.48	0	1.05	2.69	10.99
Max block size [MB]		1	Dynamic[a]	1	N/A[b]	1	2	Dynamic[c]

a Variable block size based on gas limit.
b Does not have blocks.
c Variable block size based on last 100 blocks.

blockchain deployment. For example, Bitcoin can handle up to 7 TPS, far from PayPal and VISA's performance, which has approximately 200 TPS and 20 k TPS, respectively [16].

Energy Consumption

Energy consumption is another important performance criterion of a consensus algorithm. It is well known that the total energy consumption for mining in the Bitcoin network can now power a whole country like Portugal, Singapore, and Czech Republic, to name a few [17]. Furthermore, the primary source of electricity that runs the Bitcoin network is from coal-fired power plants in China, which is infamous for its extreme amount of carbon emission. A study conducted in 2018 suggests that the carbon emission related to Bitcoin alone could increase global warming by 2 °C in less than three decades [18].

The high energy consumption is mainly due to the computation of the cryptographic hash functions such as SHA-256 (e.g., Bitcoin), Ethash (e.g., Ethereum), and many other hash functions used by the PoW algorithms. As the difficulty of the PoW algorithm continues to increase, so does the energy consumption, and it is pivotal that future consensus algorithms focus on energy efficiency as a top objective.

Security Criteria

The blockchains have various cybersecurity attack vectors that can threaten any given consensus algorithms. Naturally, one could think of diverse attacks and vulnerabilities when given a particular consensus algorithm. However, in this paper, we present only well-known attacks that can be commonly applied to most of consensus algorithms. Namely, adversary tolerance, double-spending prevention, and Sybil protection resistance will be explored in detail.

Fault (or Adversary) Tolerance

To begin with, fault tolerance typically refers to crash fault tolerance in which nodes or members of a network fail and become offline until they are brought back online. This is different from becoming compromised and sending fraudulent transactions. When f number of nodes or members has crashed, the network requires $2f + 1$ participants or a quorum of $f + 1$ to be crash fault tolerant.

If the nodes are subverted and send fraudulent transactions to the network, this is called Byzantine behavior [19], and their consensus algorithm must be Byzantine fault tolerant. One of the well-known algorithms that can achieve consensus in this attack is Practical Byzantine Fault Tolerant (PBFT) [20], which is capable of handling up to f Byzantine nodes with $3f + 1$ total nodes.

By contrast, when we take proof-based consensus algorithms into account, the term "fault tolerance" refers to the percentage of total network resources that need to assure consensus. When an adversary is able to control more than 50% of a network's computing power, it could maliciously alter or control the consensus process to launch an attack (e.g., double spending). Hence, the term *51% attack* is widely used as it

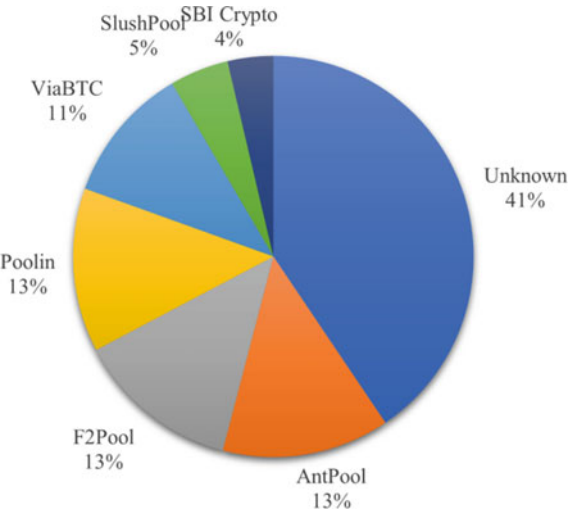

Fig. 2 A hash rate distribution of Bitcoin mining pools in February 2022 is shown

was also discussed in the original Bitcoin paper [21]. A selfish mining strategy shows that an adversary with less than 50% power could withhold block and increase its profitability without affecting the safety or liveness of a blockchain network [22]. It is also worth mentioning that in theory 51% attack is unavoidable, and adversaries with mining pools could always collude with each other. For example, Fig. 2 shows that the top five mining pools of Bitcoin exceed 51% of the total hash rate of the whole network as of early 2022 [23].

Double-Spending Prevention

In a double-spending attack, the adversary creates a typical transaction that reaches consensus, and then the adversary creates a fork with a conflicting transaction to revert the prior transaction. A successful double-spending attack will allow the adversary to spend the same coin more than once, hence double.

However, in order to push the fork with the malformed transaction to the main chain, the adversary must have already broken the adversary tolerance. For example, in the Bitcoin PoW algorithm, the adversary should have at least 51% computing power of the entire network to ensure a successful fork through faster block creation.

Consensus algorithms try to mitigate this attack by introducing a *confirmation count*, which is incremented when a block of the transaction of interest is followed by another block. Subsequent blocks increase the number of confirmations, which in turn increases the probability of the transaction validity. This is also referred to as *Probabilistic finality*. In essence, the probability of an invalid fork that reverts a prior transaction decreases exponentially as more blocks are appended to the chain [13].

Table 3 presents the number of confirmations required and the average validation time of well-known cryptocurrencies. Notice that the number of confirmations multiplied by average latency or block generation time from Table 2 is the average validation time.

Table 3 Number of confirmations and average validation time of selected cryptocurrencies

Cryptocurrency	Number of confirmations	Average validation time [Min]
Bitcoin	6	60
Ethereum	30	6
Litecoin	12	30
Ripple	N/A	N/A
Dogecoin	20	20
DASH	6	15
Monero	15	30

Sybil Protection

A Sybil attack [24] takes place when an adversary attempts to control the network by duplicating fraudulent identities. Within the blockchain, a Sybil attack will be an attempt to create and possess as many nodes or members of the network in order to influence the consensus algorithm. A successful attack may grant the adversary the higher voting power in consensus algorithms that utilize a voting process (e.g., DPoS, PBFT), or enable network layer attacks targeting peer discovery and block broadcasts [25].

To prevent Sybil attacks, consensus algorithms could use combinations of methods from increasing the cost of creating identities, requiring second-channel authentication, or two-step verification for identities [26, 27].

Decentralization Criteria

We identified two factors that can qualitatively evaluate the consensus algorithms decentralization: permission and censorship resistance.

Permission

Depending on blockchains, a node may need a permission to participate in reaching a consensus by validating transactions and creating blocks. A *permissionless* consensus algorithm will allow any anonymous node to participate in the consensus process. While a *permissioned* consensus algorithm will only allow authenticated nodes to participate in the consensus process. This is not to be confused with the concept of *public* and *private* blockchain, where permission refers to the anonymity of miners and validators, while public and private refers to the anonymity of all the nodes participating in the blockchain network. For example, the Bitcoin and the Ethereum network is a public permissionless network. A blockchain-based voting system should be a public but permissioned network, so that the voters remain anonymous while the validators are authenticated to be trustworthy.

The number of nodes in the permissionless network will be large compared to that of a permissioned network, and to mitigate attack vectors such as 51% attacks

and Sybil attacks, proof-based consensus algorithms are used. As a result, all validators must spend energy and resources to prove its contribution to the network by participating in the consensus process. In general, permissioned networks are more centralized compared to the permissionless network.

Censorship Resistance

Censorship resistance refers to the network's property that assures any node to freely make transactions as long as they follow the rules of the consensus algorithm and the blockchain network. With traditional finance institutions, some intermediaries would censor transactions that it deemed suspicious or undesirable, justified to prevent financial crimes. Also, if probabilistic finality is achieved, a transaction recorded on the blockchain is technically irreversible, also commonly known as *immutable*, providing further censorship resistance [28].

3 Consensus Algorithms

Performing a consensus algorithm in a blockchain network is a non-trivial process. A newly broadcasted transaction is first verified by a verifying node and added to its candidate block. Similarly, the candidate block will be verified or voted by other nodes in the network before being added to the chain. We would like to emphasize that once a transaction is included in the chain, it is not feasible to modify or delete them due to blockchains' immutability.

The consensus algorithms that will be discussed in this article can be classified into two categories: proof-based and vote-based. A similar distinction between the two categories can be made using the permission criterion. In permissionless blockchains, any nodes are free to join and leave the network, and their behaviors are unpredictable. Therefore, a permissionless blockchain typically relies on a *proving* mechanism that appreciates verifying nodes' contribution toward the network. This usually involves rewards, which incentivizes the nodes to participate in the consensus process. In contrast, a vote-based consensus algorithm does not require contribution or *proof* from the node participating in the consensus process since the participants are *permissioned* in advance, and their participation list is managed. Thus, vote-based consensus algorithms can be adopted in *non-incentivized* blockchain networks and are well-suited for private blockchains and non-cryptocurrency applications**.**

Note that the voting process does not always go along with vote-based consensus algorithms since it can be used in proof-based consensus algorithms (e.g., DPoS, BFT PoS). The difference between the above two categories lies in whether the verifying node of the consensus algorithm is required to provide a *proof* (e.g., computation, stake) to the network to participate in the consensus process.

Figure 3 shows the classification of consensus algorithms that will be discussed in this section.

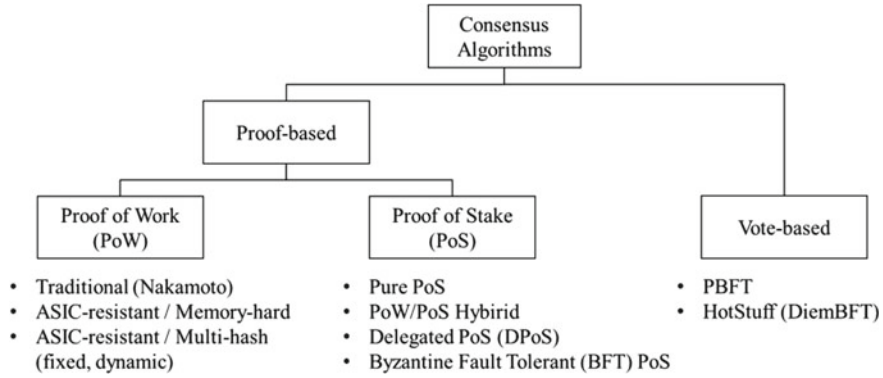

Fig. 3 A classification of consensus algorithms surveyed in this article is summarized

3.1 Proof-Based Consensus Algorithms

The original Proof of Work (PoW) of Bitcoin is the most popular proof-based consensus algorithm to date. As mentioned earlier, the basic concept of a proof-based consensus algorithm is that a participating node of the consensus process performs or provides a sufficient proof to append a new block and is rewarded. Depending on the method of a proof, the proof-based consensus algorithms can be further divided into proof of work (computation), proof of stake (currency locked in escrow), proof of activity (transaction participation), proof of research (Berkeley Open Infrastructure for Network Computing contribution), and so on.

In this article, we will focus on the two major proof-based consensus algorithms, PoW and PoS.

3.2 Proof of Work (PoW)

The Proof-of-Work (PoW) algorithm usually requires a proving node and a verifying node. The proving node performs a resource-intensive computational task to find a solution to a problem of a certain difficulty level. The result is then presented to the verifier who spends a significantly less resource for validation compared to the prover. The asymmetry and the excessive amount of resources required for the prover serves two notable purposes:

1. It mitigates Sybil attacks at the consensus level. Launching a Sybil attack involves the adversary creating multiple fraudulent identities. However, by design, the amount of computational resources (e.g., hash rate) is important for the PoW algorithm, not the number of nodes. Note that Sybil attacks in the network layer are still a vulnerability, but this article focuses on the consensus layer.

2. The workload itself becomes a safeguard against forks and double-spending attacks. The length of the chain is almost proportional to the amount of resources spent mining the blocks. If the adversary wants to modify a transaction from the past, he will first have to acquire more than 51% of computing resource within the network, fork a new chain starting from the target block, and exhaustively mine the blocks until the new chain becomes the longest.

There are two major sub-categories of PoW algorithms: traditional PoW and ASIC-resistant PoW, which are to be explored in the following.

Traditional PoW

Traditional PoW algorithm employs computational tasks that heavily exploit either the CPU or the GPU with little dependency on the system memory size. Another critical characteristic of computation-bound PoW is that the computation can easily be implemented on an ASIC, which leads to mining farms and mining pools, counteracting the notion of decentralized consensus.

The earliest idea of computation-bound PoW algorithm dates to 2002, an anti-spam system called HashCash [29]. This system requires the sender of an email to generate a SHA-1 hash value with at least 20 bits of leading zeros. The list of inputs included the recipient's address and date alongside the random number, called a *nonce*, provided by the sender. The sender should try numerous proof attempts to meet the leading zeros requirement, while the verification by the recipient is relatively trivial.

Bitcoin's PoW algorithm is based on the HashCash's PoW algorithm, but is modified to use SHA-256d (SHA-256 performed twice) instead of SHA-1. Fundamentally, provers of the Bitcoin network are trying to find a 256-bit nonce that, when hashed with the block, will have outcome which is smaller than the *difficulty value*. Recently, Bitcoin's hash rate surpassed 150Exahash/s (Exa $= 10^{18}$) [15], which means in order to append a block, an average of 90×10^{21} nonces is tried. When an appropriate nonce is found and approved by the verifiers, the prover receives a block reward of 6.25 BTC as of March 2021. The process of finding the nonce is known as *mining*, and the proving nodes are called *miners*. Figure 4 shows the process of a miner finding the nonce.

When a miner successfully finds a nonce that satisfies the difficulty, it broadcasts the mined block to the entire network. Other verifying nodes, who might also have been mining a block at the same height, verify the newly broadcasted block by checking whether all the transactions included within the block are valid, whether the previous hash value (*Prev_Hash*) matches with the hash value of the last block from their current chain, and whether the nonce value satisfies the current difficulty. If these conditions are met, the mining (or proving) nodes abandon their current task and append the new block to their chain. And then, they reselect the transactions to be included in the next block by referencing the latest block, form a new transactions list and the next block header, and start over the nonce calculation.

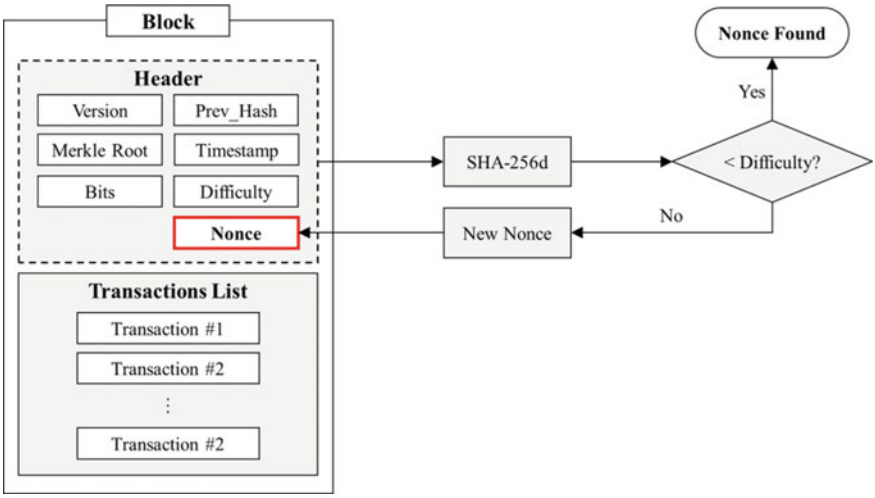

Fig. 4 A process of finding the nonce in Bitcoin PoW is shown

Once in a while, due to the worldwide scale of the Bitcoin network and large pool of miners, there may be more than one new valid blocks broadcasted throughout the network at the same height. As a result, the network is divided based on which one of the legitimate blocks the node received first as shown in Fig. 5. Here, the two black-colored nodes each successfully mined two new blocks N and N‘ that are appended to the most recent block $N-1$. Due to the size of the network and broadcast latency, the nodes in Group 2 received the block N‘ faster than the block N. Hence, the nodes in Group 2 appended the block $N-1$ with block N‘, and the nodes in Group 1 with block N. Now there are two concurrently valid chains in the network and the global consensus is now broken. This situation is called a *fork*.

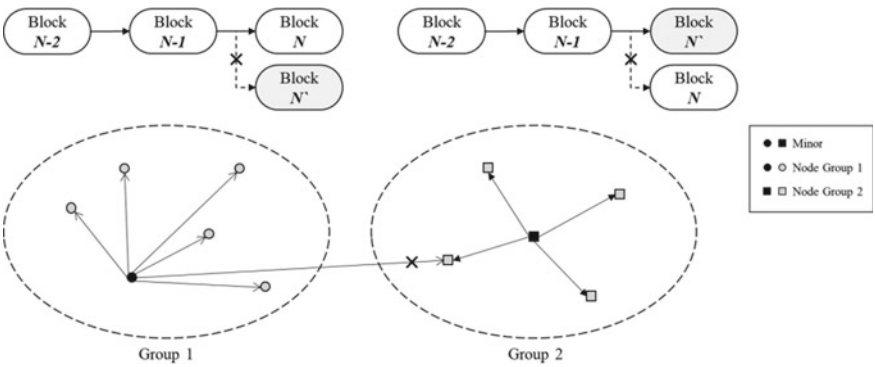

Fig. 5 A fork in blockchain takes place at block height N

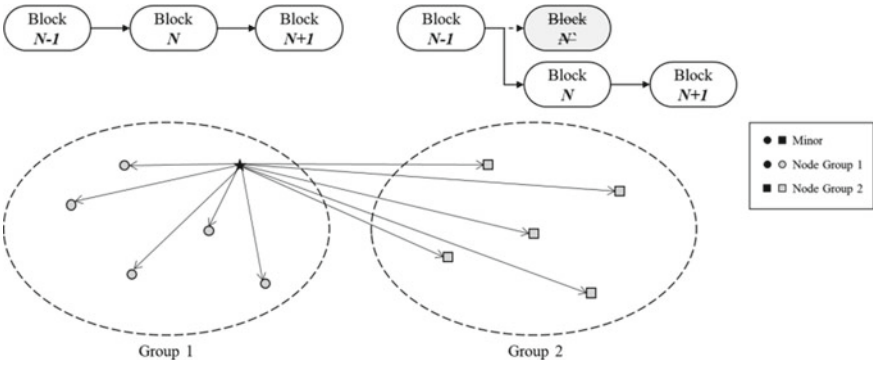

Fig. 6 How a fork is resolved through the longest chain rule is illustrated

To overcome this issue and achieve consensus, first the nodes proceed to mine new blocks based on their version (or branch) of the chain. For example, the next block $N + 1$ is mined by a single miner (highlighted in star shape in Fig. 6). Block $N + 1$ gets broadcasted to all the nodes of the network. The nodes of Group 1 share the same block N as the current block and append it with the newly mined block $N + 1$. However, the nodes of Group 2 do not recognize the previous hash of block $N + 1$, which is N, and are unable to append block $N + 1$ to N'. Now, the two branches of the fork have a length difference of a block.

The chain length indicates the amount of computational resource put into creating the blocks and managing the chain. When given multiple branches of a chain, the branch with the longest chain is allocated with the most computational power, i.e., hash rate. Thus, the nodes of Group 2 abandon the former consensus based on block N' and select $N \rightarrow N + 1$ as their new chain. This is known as the *longest chain rule*, and this helps resolve forks and reachieve global consensus in the network.

As mentioned at the beginning of this subsection, traditional PoW algorithms are subject to ASIC. Specifically, this means that the hash function used in traditional PoW algorithms such as SHA-256 in Bitcoin is easy to implement using an ASIC. A study in 2018 showed that an ASIC-based system outperformed a general-purpose computing system (e.g., PC with high-end GPU) of equal power by more than 1500 times in terms of hash rate [30]. Heavy use of ASICs and the domination of mining pools centralized the consensus layer of traditional PoW algorithms, violating the original intention of achieving decentralized consensus through distributed computing resources. Also, the overheated competition of mining has raised concerns about energy/resource wastes and environmental implications.

ASIC-Resistant PoW

To overcome issues regarding expediting traditional PoW algorithms in ASIC machines, ASIC-resistant PoW algorithms are gaining more and more interests in

the community. In this article, we classify ASIC-resistant PoWs into two categories: memory-hard PoWs and multi-hash PoWs.

Memory-hard PoW: While ASICs have an advantage in hash rate, they are also limited by memory access latency, bandwidth, and memory size. Memory-hard PoW algorithms restrict ASICs from having a performance advantage over general-purpose computing systems by requiring the use of random pieces of data from a large dataset. The dataset should be too large to be stored in the on-chip memory of an ASIC chip. Nonetheless, the Ethash of Ethereum [31], an ASIC-resistant memory-hard PoW, was broken in 2018 by Bitmain's Antminer E3 [32], and now there are ASICs available for memory-hard PoW as well. Still, unlike traditional PoW schemes where mining can heavily leverage ASICs, GPU-based general-purpose computing systems are still dominant in Ethereum [33].

The first memory-hard PoW algorithm appeared before cryptocurrencies in the form of password-based Key Derivation Function (KDF) called Scrypt in 2009 [34, 35]. It was later reinstated as a PoW algorithm of Tenebrix (no longer active), Litecoin [36], Dogecoin [37], and few other cryptocurrencies. Scrypt aims to resist against custom hardware such as GPU, FPGA, and ASIC, and hence turns out to be a CPU-friendly PoW. Both memory-hardness and CPU-friendliness are achieved by read-many, write-few memory access patterns of PoW algorithms. However, ASIC engineers reduced the memory size by 1/8 with a 3.5× additional logic calculation. Thus, Scrypt's memory-hardness broke, and ASIC efficiency gains were 300,000 times over a CPU [38].

Ethash, also known as the Dagger-Hashimoto algorithm, is a memory-hard consensus algorithm of Ethereum [31], the second largest cryptocurrency in terms of market capitalization. Noticeably, during the mining process, Ethash requires data from a Directed Acyclic Graph (DAG) dataset that is over 4 GB in size, as shown on the right side of Fig. 7. To construct a DAG, first, the *seed hash* must be hashed N consecutive times using the keccak-256 [39] hashing algorithm, an early version of SHA-3, where N is the current *epoch* of the DAG. Next, the current *seed hash* is used to calculate a *pseudorandom cache* that is again used to generate the actual DAG dataset. After every 30,000 blocks (approximately 5 days), a new *mining season* begins, and the epoch is increased by one, consequently updating the seed hash, pseudorandom cache, and the DAG dataset. By design, the size of the pseudorandom cache and the DAG dataset is configured to increase linearly by each epoch, starting from 16 MB for the cache and 1 GB for the DAG dataset. In more than 20 years, Ethash will reach epoch #2048, where the cache size reaches 285 MB and the DAG dataset reaches 18.2 GB.

The left side of Fig. 7 depicts the mining process using the generated DAG. Like Bitcoin's PoW algorithm, a random nonce is selected, and after several hashes and mixes with input data from various stages, the final output is compared with the difficulty value to determine the result of the nonce. Every *mixing* operation requires a value from a random address of the DAG, involving memory access and enough storage for the DAG. Although the hashing and mixing logic can be built using an ASIC, the memory bandwidth serves as a bottleneck balancing the end performance between an ASIC-based system and a general-purpose computing system.

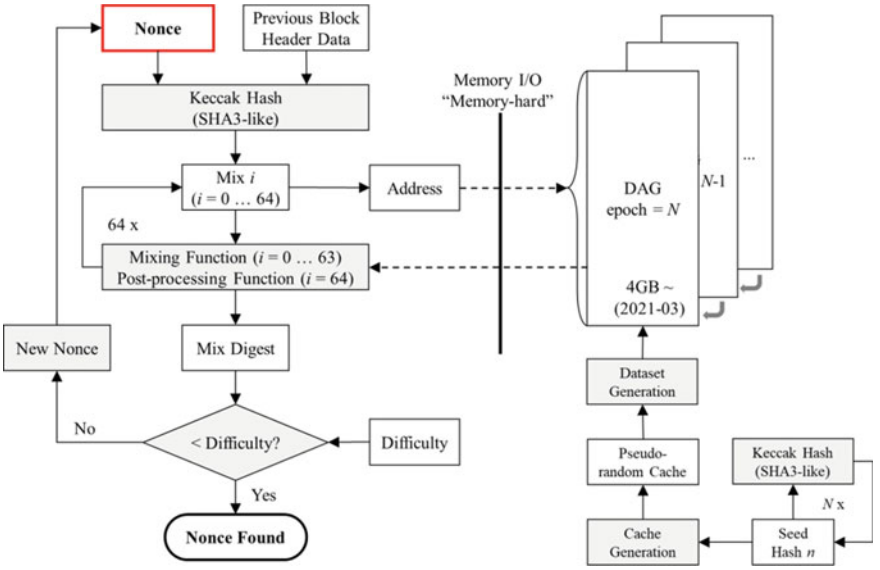

Fig. 7 A process of finding the nonce in Ethash is shown

Multi-hash PoW: Another group of ASIC-resistant PoW algorithms is *multi-hash* PoWs or *chained* PoWs. As their name suggests, these consensus algorithms do not rely on a single hash function but multiple hash functions during the mining process. There are a few ways to do this. First, we can define a sequence of hash functions forming a fixed hash chain. An example of fixed multi-hash PoW algorithm, X11 [39], used in the DASH blockchain, is presented in Fig. 8. Here, the algorithm uses 11 hash functions: Blake, Blue Midnight With (BMW), Grøstl, JH, keccak, Skein, Luffa, CubeHash, SHAVite, SIMD, and Echo. Keccak is the winner of the SHA-3 [40] open competition, and others in the list include those advanced to the second round or the final round. There are also the variants of X11 such as X13, X14, X15, and X17 which utilize more hash functions in their sequence of hashes.

Another way to implement multi-hash PoW algorithms is using a dynamic sequence of hash functions through several methods. One is a fully variable sequence of hash functions where the sequence is permuted by a random value based on block hash or timestamp. Alternatively, a partially variable sequence with randomly selected hash functions in between a fixed sequence is also possible. An example of a fully variable sequence is X16R [41] which is shown in Fig. 9. The X16R algorithm constructs the sequence of hash functions based on the last 8 bytes of the previous block's hash. Each hashing algorithm is mapped with a hexadecimal value, and the last 8 bytes, i.e., 16 hexadecimal values, determine the permutation of hash functions for the current block. Also, note that repeated use of a hash function is possible when the two consecutive hexadecimal values are the same. Nonetheless, we can safely assume that the use of hash functions would be balanced due to the randomness property of cryptographic hash functions.

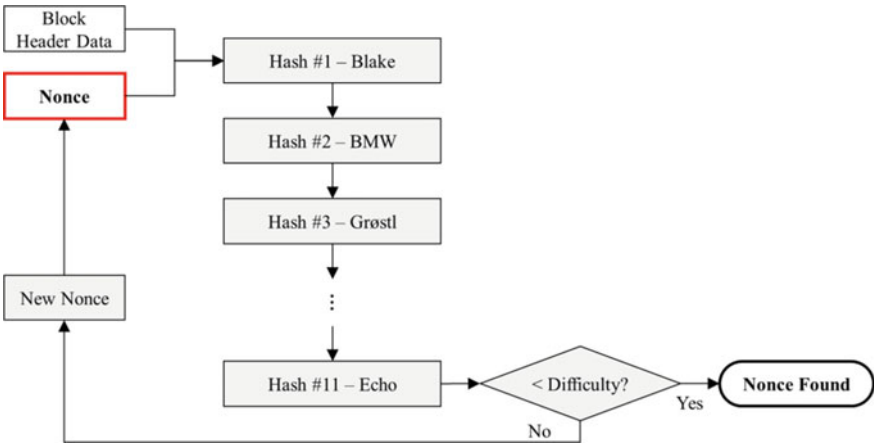

Fig. 8 How to find a nonce in fixed multi-hash PoW, X11, is depicted

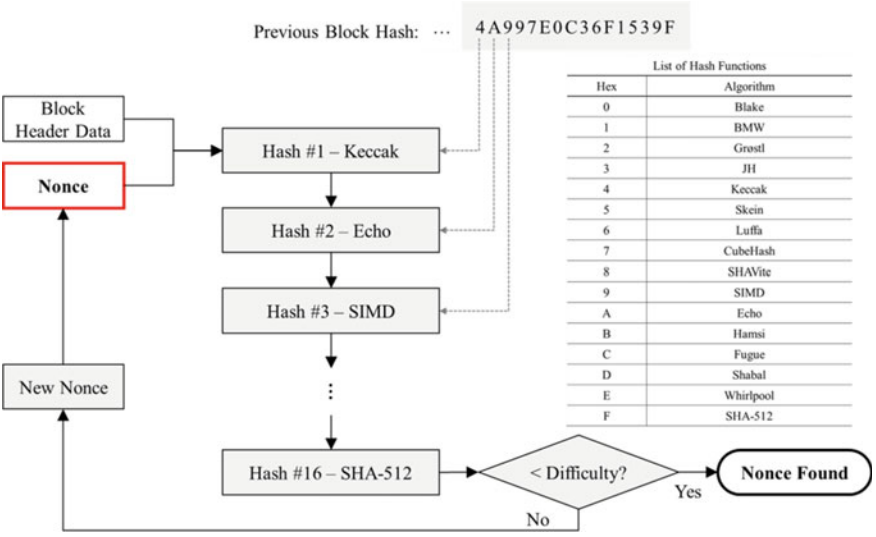

Fig. 9 How to find a nonce in a variable multi-hash PoW, X16R, is shown

Multi-hash PoW algorithms were first thought to be ASIC-resistant since they required implementing multiple hash functions in hardware. Whereas in a general-purpose computing system, it was a simple matter of switching codes in software to execute the different hash functions. This becomes more apparent when we consider variable multi-hash PoW algorithms. It is not profitable to design an ASIC chip that supports all possible sequences of hash functions, for example, 16^{16} in the case of X16R (since it allows repeated usage). However, this is true only if we assume that hash modules in an ASIC must be connected in a fixed pipeline. It is reported that

supporting dynamic combinations of hash functions in hardware are possible with some overhead [30]. As to the fixed multi-hash PoWs, it took 2 years for an ASIC targeted for X11 in DASH to be commercially available. We believe developing an ASIC for a variable sequence of hash functions is a matter of time.

Programmatic PoW: Another ASIC-resistant PoW algorithm known as *programmatic* PoW achieves ASIC-resistance by using a large pool of mathematical functions or randomly generated programs [42, 43]. However, there is no PoW algorithm of this category used in an existing blockchain application as of today.

3.3 Proof of Stake (PoS)

To overcome the downside of PoW algorithms, such as mining centralization, energy waste, and low TPS, a new consensus algorithm called Proof of Stake (PoS) is gaining a momentum [44, 45]. A PoS algorithm attempts to validate transactions and achieves consensus in the network without the heavy computational tasks like PoW algorithms. Instead of *working*, a verifying node has to lock its *stake*, a proportion of its wealth on the network.

In the PoS algorithm, stakeholders lock their stakes in escrow and become eligible to participate in transaction validation and block creation. Unlike the PoW algorithm, the stakeholders in PoS do not mine new coins by validating transactions and creating new blocks. Instead, they collect the transaction fees or interests proportional to their stakes as rewards. Here, the term *minting* or *forging* is commonly used instead of *mining,* and the stakeholder who creates a new block is referred to as a *validator*, *forger*, or *minter*, not a *miner*. If the validator is found to behave maliciously, the escrowed stake and possible reward will be confiscated, which serves as an economic disincentive for adversarial stakeholders in PoS algorithms.

The advantages of PoS algorithms over PoW algorithms are as follows: decentralization, energy efficiency, and high throughput.

1. The high performance of ASICs has centralized the mining of PoW-based applications. Due to the economy of scale, miners can expect an exponential increment of computational resources (e.g., hash power) and the ensuing rewards when investing in PoW-targeted mining equipments. On the contrary, the expected gain in network control in PoS is directly proportional to the amount of additional stake that the validator put in escrow. Although we cannot claim that PoS algorithms are immune to centralization, it will likely be more decentralized compared to PoW-based networks.
2. As discussed in section "Performance Criteria", Bitcoin miners consume more energy than countries like Portugal, Singapore, and the Czech Republic. PoW algorithms are spending natural resources too much to secure the immutability of Bitcoin transactions. PoS algorithms do not require validators to solve resource-intensive computational tasks when forging blocks, thus they can be made highly

energy efficient. Any blockchain applications that are based on PoS will be more sustainable in the long run.

3. PoW algorithms suffer from low TPS and high block generation times. For instance, a payment made by a PoW-based cryptocurrency normally requires at least one block confirmation to be approved. What is worse, if a burst of transactions suddenly floods the network, PoW-based cryptocurrencies may not handle them timely due to the limited TPS. On the other hand, PoS algorithms offer relatively higher TPS and shorter block generation time by using faster agreement mechanisms for accepting a new block to the chain, ensuring higher throughput compared to PoW algorithms.

PoS algorithms can be classified into three major categories: chain-based PoS, Delegated PoS (DPoS), and Byzantine Fault-Tolerant PoS (BFT PoS). These are classified based on their difference in terms of validator selection, block creation, and agreement.

Chain-Based PoS

A chain-based PoS algorithm mimics PoW mechanics by featuring validators competing one another to *mint* (not *mine*) a new block. Instead of going through the resource-intensive process of finding a valid nonce, the validators are pseudo-randomly selected to mint a block based on their stakes. Like PoW algorithms, the chain-based PoS algorithm's finality is not achieved at the time of block creation. Forks could happen when multiple validators mint the new block simultaneously across the network. Thus, it should have a mechanism to converge the branches of the chain.

This article classifies chain-based PoW algorithms into two sub-categories depending on how to select the validator: Pure PoS with randomized probabilistic selection and PoW/PoS Hybrid with coin-age-based selection.

Pure PoS: A chain with a pure PoS algorithm consists of blocks only minted by validators. Recall that in PoW algorithms, the chain's length is almost proportional to the amount of work put into the chain, and thus the network converges to the longest chain if a fork happens. Similarly, in PoS, the chain's length indicates the sum of stakes escrowed by validators, and the nodes are expected to follow the longest chain.

Nxt [46] is a pure PoS-based cryptocurrency that utilizes a randomized probabilistic function in selecting a validator. When a new block is added to the chain, an active Nxt account (or node) i performs a sequence of hashes using its public key and the latest block's signature to generate an *account hit* value H_i. The use of an individual account's public key and the signature value of an independent block provide a pseudo-randomness to H_i. Next, the accounts calculate their target values T_i.

$$T_i = T_b \times t \times B_i$$

T_i: target value of account *i,*

T_b: base target value,

t: seconds passed since the last block was generated, and

B_i: effective balance (stake) of account *i.*

As time goes by, the target values of active accounts increase, and the first account with a target value larger than its account hit value, $H_i < T_i$, gains the right to mint the next block. Notice that T_b and t are shared within the network, and B_i determines how fast the target value reaches H_i. The randomness of H_i allows other accounts to be selected as the validator, although the probability diminishes as one's stake gets smaller.

The new block gets broadcasted to the network, and every account updates its account hit and target values to repeat the process from $t = 0$.

PoW/PoS Hybrid: As its name suggests, PoW/PoS hybrid algorithms are a mixture of both PoW and PoS. PeerCoin (PPCoin) [45] is the first variant of this category, but interestingly, it is also the first PoS algorithm to be used in cryptocurrencies.

PeerCoin recognizes two kinds of blocks: PoW blocks and PoS blocks. PoW blocks are similar to the blocks of Bitcoin, mined by miners competing with each other to find a valid block with a nonce satisfying the target difficulty. Their purpose is to increase the net supply of the coins within the network. PoS blocks, on the other hand, are minted by minters (or validators), who are competing against one another based on their stakes for a target difficulty with far less computation. Thus, the minters are in charge of performing the consensus by adding PoS blocks [47]. From a broader view, miners and minters are competing against each other as two groups since the new block could be either a PoW block or a PoS block.

In selecting the minter of the next PoS block, PeerCoin employs the concept of *coin-age*. Coin-age is the product of the amount of token/coin in an account and its holding period. For example, if Alice has held 100 coins in her wallet for 100 days and Bob has just received 500 coins. In terms of coin-age, Alice has accumulated 10,000 coin-age (days) and Bob 500 coin-age (days), making Alice a more prominent stakeholder of the network at the time.

The target difficulty of a valid PoS block gets easier as more coin-age is accumulated, and if a minter successfully mints a block, she burns all the coin-age by including a transaction of paying the stake to herself. This ensures that the stakes used in minting the block are not used until she accumulates a certain amount of coin-age, enabling every participant to mint blocks with fairness long term.

A significant issue of chain-based PoS is its vulnerability to Nothing-at-Stake (NAS) attacks. Whenever there is a fork in the chain, the nodes working on the next block should decide which branch of the fork it will work on. For PoW algorithms, this means that a miner should invest its computational resource (and energy) in finding the nonce. Thus, it is rational for the miner to select only one branch. The PoW algorithm would give a penalty for working on multiple branches. However,

for PoS algorithms, it costs little for a minter to work on multiple branches and, in fact, it increases the possibility of minting the next block, which motivates the minter to do so.

One solution to solve NAS, proposed by another chain-based PoW/PoS hybrid algorithm known as Casper the Friendly Finality Gadget (Casper FFG), is penalizing (or *slashing*) the stake of a validator who works on two conflicting blocks.

Delegated PoS (DPoS)

Despite the similarity of terminology, DPoS, is substantially different from the original PoS in the sense that the network allows stakeholders to *delegate* their voting power to a particular participant who can then stake on behalf of her voters [48]. The rewards (i.e., transaction fees and interests) earned by delegates (also called witnesses) can trickle down to voters with a small amount of stakes, allowing them to participate in the consensus process for incentives.

Unfortunately, by design, DPoS centralizes the consensus layer since a small number of selected delegates manage the chain. However, this allows higher throughput and better scalability since not all the nodes have to participate in block creation and chain management.

Figure 10 shows a simplified version of DPoS. First, the stakeholders who possess any amount of stakes give their votes to delegate candidates. The more stake a stakeholder has, the more voting power it can exert. In the end, delegate candidates are ranked based on their stakes, and a predefined number (e.g., EOS: 21, Ark: 51, Lisk: 101) of candidates are elected as the group of delegates. The election is done periodically, and the stakeholders can freely reallocate their voting power to different candidates, and new delegates will be chosen. Also, the system can define a backup pool of delegates so that when a delegate in the main group is unable to participate in the block creation, it will be replaced by a backup delegate.

Within the group of delegates, the witnesses equally take turns (e.g., round-robin) creating and proposing new blocks. Thus, it is essential for the witness to always be online while she is working in the group. When a witness forges a new block, it can be added to the chain directly or go through a voting phase, as shown in Fig. 10. If the proposed block gets more than 2/3 of the votes, it is then appended to the current chain, and the next witness starts creating the next block. The rewards earned from participating as a delegate will be redistributed to the stakeholders who voted for the delegate. (Note that a portion of the rewards can be spent on other causes such as the platform development or charity.) This redistribution plan can also affect the stakeholders in placing their votes on candidates.

Other cryptocurrencies that utilize DPoS include EOS, BitShares, Ark, Lisk, and Tezos.

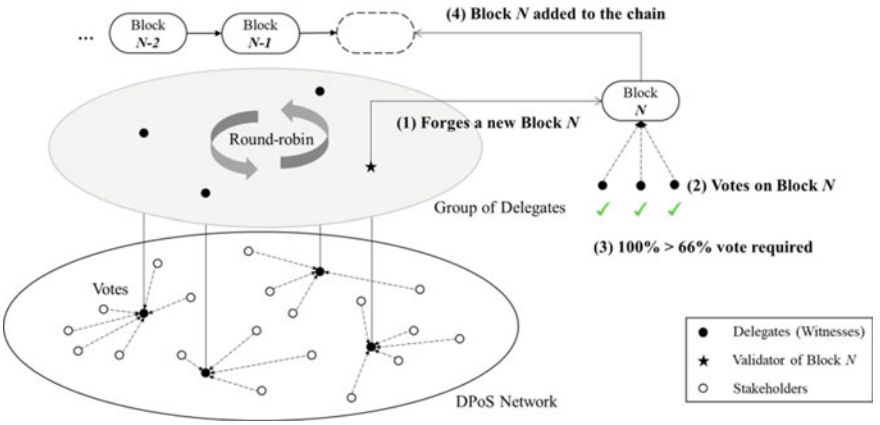

Fig. 10 How to reach consensus in DPoS is illustrated

Byzantine Fault-Tolerant PoS (BFT PoS)

BFT PoS is based on a round-based voting process. Unlike PoW or chain-based PoS algorithms where a consensus is reached through the chain's length, BFT PoS achieves consensus for every voting round whenever a block is forged.

Tendermint [49, 50] is the first BFT PoS that showed BFT consensus could be achieved with PoS. We focus on the Tendermint Core consensus engine that works as a round-based voting mechanism.

Similar to DPoS, the network requires a set of validators who maintain the chain and take turns proposing and committing new blocks for every block height. The validator responsible for the block proposal in a round is called a *proposer*, and other validators validate the proposed block and place votes. The application can define the voting power of the votes cast by the validators. In other words, Tendermint Core allows both even distribution of voting power per validator and weighted voting power based on stakes. In BFT PoS, we assume that the validators are selected by stakes or through delegation like in DPoS, and that their voting power and frequency of being a proposer are also proportional to the stakes involved.

Each round of creating a new block consists of three steps with equal timeouts: *propose*, *pre-vote*, *pre-commit*, and two special steps: *commit* and *new height*, as shown in Fig. 11.

In the *propose* step, a new block is forged by the current round's proposer (chosen in round-robin fashion with a weighted probability based on the stake) and broadcasted throughout the network. If the block is valid, every validator goes to the *pre-vote* step and broadcasts a pre-vote. Else, if the block is not valid, or a validator did not receive any block proposal within the defined timeout, it sends pre-vote nil.

During the *pre-vote* step, every validator waits and listens for pre-vote broadcasts and checks whether more than two-third of the voting power has been achieved on

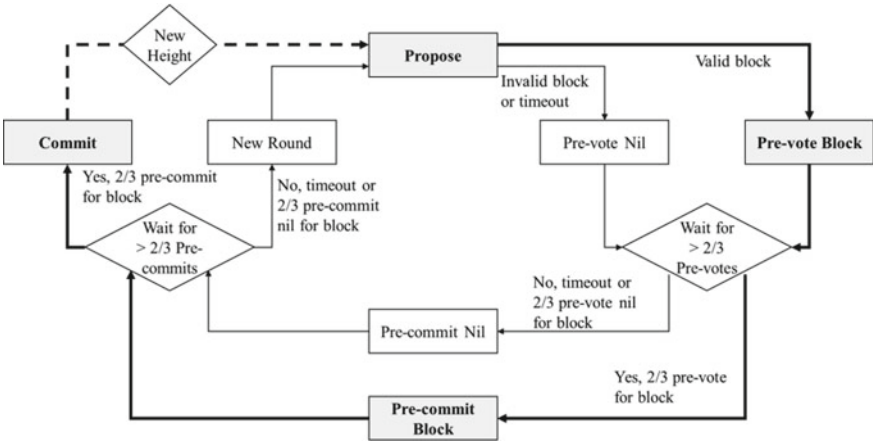

Fig. 11 A simplified state machine of Tendermint [50] is shown

pre-vote. If so, every validator moves on to the *pre-commit* step, broadcasts a pre-commit for that block, and *locks* itself to that block. On the contrary, if the validator did not receive two-third of the voting power on pre-vote until timeout, or received more than two-third of the voting power placed on pre-vote nil, it sends pre-commit nil. Here, we mentioned the concept of locking onto a block. Once a validator is locked on a block, it always sends a pre-vote for that block in future rounds of the same chain height. Also, it can only unlock that block if it receives a newer block of the same chain height with more than two-third of the voting power placed on pre-vote.

If the validator receives more than two-third of the voting power placed on pre-commit during the *pre-commit* step, it enters the *commit* step, where the block is finalized and added to the chain. The chain height is increased by one, and the next proposer starts preparing the next block. However, if two-third of the voting power is not reached during the *pre-commit* step, the next step is the *new round*. A new proposer is selected to propose a block of the same chain height. If the new proposer has locked herself to the block from the previous round, it would propose the same block, and other validators who have also locked themselves to this block would pre-vote this block.

As seen from the above sequence of steps, Tendermint relies heavily on achieving more than two-third of the voting power to achieve consensus. If more than one-third of voting power is somehow offline or showing Byzantine behaviors, the network will fall into an endless loop.

3.4 Vote-Based Consensus Algorithms

A vote-based consensus algorithm is different from a proof-based consensus algorithm in the sense that nodes responsible for creating and maintaining the chain are managed and often controlled. As mentioned in the beginning of Sect. 3, vote-based consensus algorithms mainly target permissioned and non-incentivized blockchain applications. As a result, we can approach vote-based consensus algorithms as we propose traditional methods for fault tolerance in distributed systems.

Nodes of a distributed system can suffer from crash faults (i.e., non-Byzantine faults), where they halt or disconnect from the network caused by hardware failures, broken network, or software issues. Alternatively, the nodes can maliciously forge or tamper with the information, which we refer to as Byzantine faults. Thus, the algorithms of this category can be further classified as being Crash Fault Tolerant (CFT) or Byzantine Fault Tolerant (BFT).

Current mainstream CFT algorithms include Paxos [51] and Raft [52], the latter of which is a simplified and implementation-friendly derivative of the former. CFT algorithms could not guarantee system reliability and resiliency in the presence of Byzantine faults and are thus used in a closed environment like IPFS [53] and IBM Hyperledger Fabric [54]. In this article, we focus on BFT algorithms that can tolerate both crash and Byzantine faults, providing better security and implementation flexibility.

First, we will present PBFT [20], representing a family of protocols for tolerating Byzantine faults. Various adaptations of PBFT include Redundant BFT (RBFT) [55], delegated BFT (dBFT) [56], and BFT-SMaRt [57]. Then, we will present HotStuff [58], a recent variation of PBFT that will be the base consensus algorithm for Facebook's upcoming blockchain payment system *Diem* (formerly known as *Libra*) [59].

Practical Byzantine Fault Tolerance

The original PBFT algorithm is written in the context of the Network File System (NFS), where nodes are addressed as primary and replicas, and a client's request triggers the execution of the state machine. Here, based on IBM's Hyperledger Sawtooth PBFT [60] implementation, we describe the functionality of PBFT as follows.

Normal case operation: The PBFT algorithm reaches a consensus through four phases: *pre-prepare*, *prepare*, *commit*, and *finish*, as shown in Fig. 12. In this example, Node 1 is the primary and Node 0 is a crash fault node. Because there are four nodes in this example, the network is both crash fault tolerant ($f = 1, 4 \geq 2f + 1$) and Byzantine fault tolerant ($4 \geq 3f + 1$).

1. **Pre-prepare**: The primary node creates a new block and publishes it to the network. This is followed by the *pre-prepare* broadcast, which contains the block ID, the block number, view number, and the ID of the primary. Each node will verify the received block and the *pre-prepare* message; if it is valid, it will add those to its log and move on to the prepare phase.

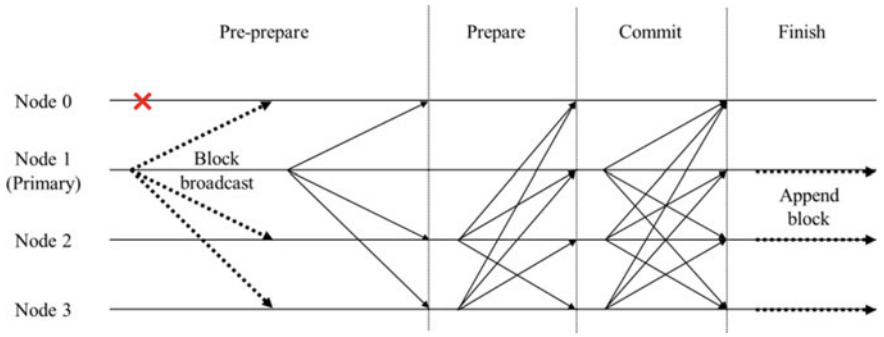

Fig. 12 A normal case operation of sawtooth PBFT [60] is illustrated

2. **Prepare**: In the prepare phase, the nodes will broadcast a *prepare* message that matches the received *pre-prepare* message to the rest of the network. Like the *pre-prepare* message, it contains the block ID and the block number, the node ID, and the view number. After sending the *prepare* message, the node will wait for *prepare* messages from other nodes. If a node has reached $2f + 1$ *prepare* messages (including the *pre-prepare* message sent by the primary) with the same block ID and number, the messages are logged, and it moves on to the commit phase.
3. **Commit**: This phase is similar to the prepare phase; the nodes broadcast a *commit* message to the network, indicating that it received and accepted *prepare* messages from two-third of the nodes or more. And again, the nodes wait until they receive $2f + 1$ matching *commit* messages from other nodes. When they reach the required number of confirmations, they move on to the finish phase. One difference between prepare phase and commit phase for the primary is that the primary is not allowed to broadcast the *prepare* message, whereas it can broadcast the *commit* message to the network after receiving $2f + 1$ *prepare* messages.
4. **Finish**: In the finish phase, the nodes will append the proposed block to their chain, increase the sequence number by one, and get ready to validate the next proposed block.

If the primary node happens to be the faulty node, there must be a way to replace the primary node to guarantee the algorithm's liveness. The primary node is concluded to be faulty in the following cases: no new block or *pre-prepare* message is sent by the primary within the timeout, a commit timeout occurs during the prepare phase, a view-change timeout occurs during the view-change operation (presented below shortly), multiple *pre-prepare* messages are received, or a *prepare* message is sent by the primary.

In any of the above cases, every node broadcasts a *view-change* message. If $f + 1$ *view-change* messages are received from other nodes (this is based on the assumption

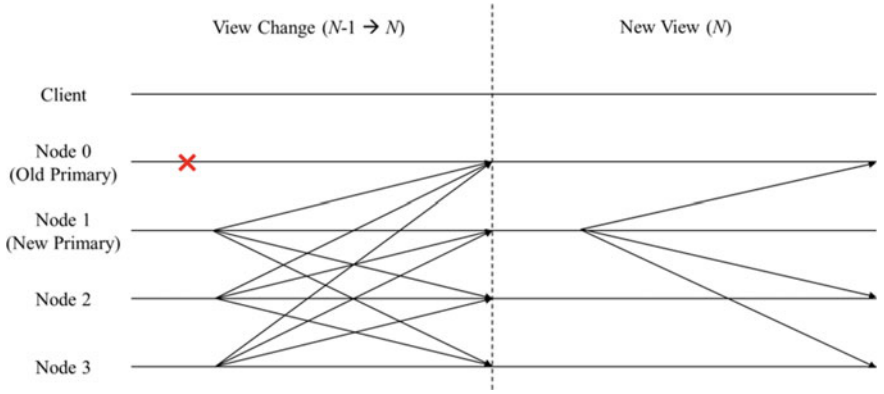

Fig. 13 A view-change operation of sawtooth PBFT is illustrated

that at most f nodes can be faulty, and receiving $f + 1$ messages indicates that regular nodes are joining the view-change), the faulty primary should be replaced by starting a view-change operation.

The view-change operation consists of two phases: view-change and new view, as shown in Fig. 13. Given the current view, $N-1$, if a node decides to start a view-change, it will broadcast a *view-change* message to the network with an updated view, N. If the primary node is indeed faulty, other non-faulty nodes will also broadcast the *view-change* messages. When the primary candidate for the new view, N, receives $2f + 1$ *view-change* messages from other nodes, it will broadcast a *new view* message for N and become the new primary, who can now start publishing blocks and send out *pre-prepare* messages.

The PBFT algorithm tolerates Byzantine faults through the broadcasts of *prepare* and *commit* messages within the network. This ensures that PBFT maintains consistency, availability, and immutability and achieves consensus. However, with the increase in the number of participating nodes, the number of broadcast messages increases quadratically, $O(n^2)$ for normal cases and $O(n^3)$ including view-change. This results in high communication overhead and degradation in performance, which makes PBFT difficult to deploy on a large scale. Many variants of PBFT try to solve this issue and achieve better scalability, one of which is HotStuff we will discuss in the following section.

HotStuff (DiemPBT)

HotStuff is a PBFT algorithm that provides linearity in the communication complexity of $O(n)$ for both normal cases and view-change cases, and view-change responsiveness. This is made possible through a few crucial design choices of HotStuff.

First, HotStuff changes the network topology from a mesh to a star, as shown in Fig. 14. As a result, the nodes no longer broadcast messages to each node; instead, it sends the message directly to the *leader* (primary in PBFT) node, significantly reducing the network's communication complexity.

Second, HotStuff is a leader-based protocol, where the view-change process is not separate but merged into the normal case process since a leader is rotated every round. Now, we will present the normal case process of basic HotStuff, as shown in Fig. 15. Note that the word *basic* is used to differentiate two types of HotStuff, which are *basic* HotStuff and *chained* HotStuff.

The basic HotStuff progresses through a series of views, which is incremented by one for every round. A round consists of the following phases: *prepare*, *pre-commit*, *commit*, and *decide*. A different leader exists for each view number, who sends out broadcasts for each phase and receives votes from other nodes. Nodes vote to a proposed branch (or block) by signing the branch with their private keys, and then

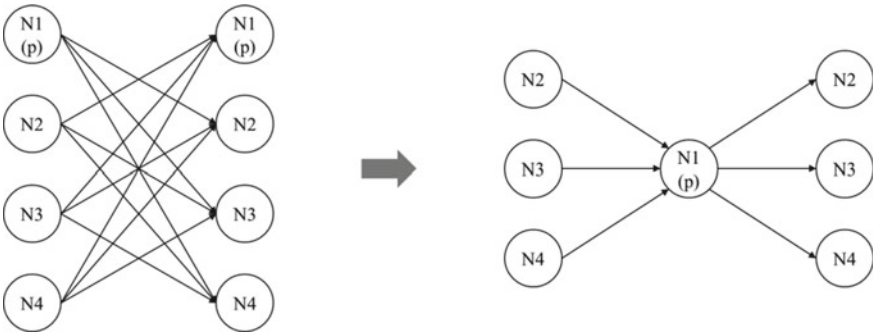

Fig. 14 While PBFT has a mesh network topology, HotStuff has a star topology. Here, p indicates primary node

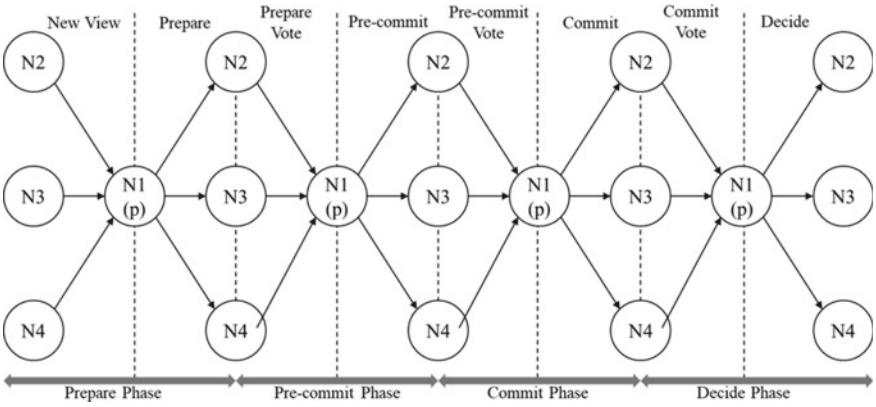

Fig. 15 The phases of basic HotStuff are shown

they send the votes to the leader. The leader collects $2f + 1$ valid votes and combines them into a *QuorumCertificate* (QC). The QC is required for each of the prepare, pre-commit, and commit phases where votes are sent by the nodes.

1. **Prepare**: The leader encapsulates the proposal into the *prepare* message and broadcasts it to the network. Every node verifies the proposal, and if accepted, returns a vote with a partial signature to the leader and moves on to the pre-commit phase.
2. **Pre-commit**: When the leader receives prepare votes from $2f + 1$ nodes for the current proposal, it combines them into a *prepareQC* and then broadcasts the *pre-commit* message to the network. The node receiving the *pre-commit* message returns a vote with a partial signature to the leader and moves on to the commit phase.
3. **Commit**: When the leader receives pre-commit votes from $2f + 1$ nodes for the pre-commit, it assembles them into a *precommitQC* and then broadcasts the *commit* message to the network. The nodes receiving the *commit* message lock their *lockedQC* to *precommitQC* and return votes with partial signatures to the leader and move on to the decide phase. The lock ensures that a consensus can be reached even if this round fails and goes through a view-change (i.e., new round).
4. **Decide**: When the leader receives commit votes from $2f + 1$ nodes for the commit, it merges them into a *commitQC* and broadcasts the *decide* message to the network. After receiving this message, the nodes execute the state transition of the proposal and move on to the next view number.

Note that a similar mechanism is repeated across the four phases; nodes vote (and sign) on a message, and the leader combines them and sends it in the following broadcasted message.

The idea of chained HotStuff is pipelining the basic HotStuff to achieve the high throughput. Specifically, the votes over the *prepare* phase are collected by the leader of the current view v_i into a *genericQC* (which is a generic version of the QCs from basic HotStuff). Then the *genericQC* is relayed to the leader of the next view v_{i+1}, delegating responsibility of starting the *pre-commit* phase. However, the leader of v_{i+1}, instead of making a *pre-commit* message, initiates a new *prepare* message and adds its proposal. This new prepare phase of v_{i+1} simultaneously serves as the pre-commit phase of the sequence started in v_i. After two additional relays, the proposal from v_i is decided as the leader of view v_{i+3} sends the *prepare* message.

The HotStuff algorithm, when compared to the PBFT, ensures higher performance due to linearity in communication complexity and the merging of normal case and view-change process into one. However, the HotStuff algorithm itself is still in its early stage, with limited evaluations done in the *testnet* for about 100 days as of March 2021. Early test results showed an average TPS of 10, with the highest value of 44 TPS [61], which is considerably less when compared to the anticipated 1000 TPS at the time of initial launch [62].

4 Evaluation

This section presents a summary of the various consensus algorithms presented in this article based on our evaluation framework.

Table 4 presents the evaluation of each criterion of the PoW algorithms (traditional, memory-hard, and multi-hash), PoS algorithms (pure PoS, PoW/PoS hybrid, DPoS, and BFT PoS), and vote-based algorithms (PBFT and HotStuff).

Throughput is not a fixed parameter that is unique to a specific category of algorithms. Even if two cryptocurrencies use the same consensus algorithm, the throughput will vary based on their latency, block size configuration, and even network size. For example, DASH started in 2014 with a block size of 1 MB, like Bitcoin. At first, DASH was able to make 28 transactions per second, max. In 2016, DASH decided (by a decentralized vote) to increase the block size to 2 MB. As a result, the current TPS of DASH tops at 56.

Overall, PoW algorithms have a relatively low throughput of one or two digit TPS. This is because they employ resource-intensive computations in reaching a consensus, which also results in their high energy consumptions. In contrast, PoS algorithms and vote-based algorithms have higher throughput of over three-digit TPS and low energy consumption due to their less computations and a fixed set of validators (reducing the accumulated energy consumption of the network).

Scalability can be evaluated in three aspects: the number of regular clients, the number of block validators, and the maximum throughput. First, a regular client who does not participate in the consensus process has minimal to no impact on the communication volume and the throughput. All consensus algorithms scale well to clients [63]. However, the situation changes a little as the number of block validators increases. Non-BFT algorithms offer great scalability due to simple communication protocols with linear communication complexity. However, BFT algorithms have several rounds/phases with a broadcast message from every participating node. This results in high communication overhead and degradation of the performance of BFT algorithms, which limits their scalability significantly. In our survey of BFT-related algorithms, PBFT, which has no predefined group of validators (i.e., primaries), offers low scalability. However, BFT PoS, DPoS, and HotStuff, each of which has a group of delegated validators, offer much better scalability. The last aspect is the throughput. If we take Bitcoin as an example, the block size limit is 1 MB, and each transaction's size is around 256 bytes on average. Thus, approximately 4,000 transactions can be contained on a block which is mined every 10 min. This leads to the maximum throughput of approximately 7 TPS. As noted earlier (Section "Performance Criteria"), reducing the latency or enlarging the block size could increase the throughput. Nevertheless, chain-based algorithms tend to achieve a TPS of under 100. On the other hand, vote-based algorithms have much higher TPS that promises better scalability, provided that their communication complexity is resolved.

In terms of fault (adversary) tolerance, chain-based algorithms all have $2f + 1$ tolerance since an adversary requires 51% of hash rate or stake in the network to

Table 4 Representative consensus algorithms are evaluated in terms of comprehensive criteria

		Performance					Security			Decentralization	
		Throughput					Fault		Double		
	Consensus algorithm	TPS	Latency	Block size	Scalability	Energy consumption	(Adversary) tolerance	Sybil protection	Spending prevention	Permissionless	Censorship Resistance
Proof of Work	Traditional[a] (Nakamoto)	5–7	1 Om	1 MB	High	High	$2f + 1$	O	X	O	High
	ASIC-resistant/Memory-hard[b] (Ethash)	13–15 (Max 30)	15 s	Dynamic	High	High	$2f + 1$	O	X	O	High
	ASIC-resistant/Multi-hash[c] (X-series)	−56 (Max)	2 m 30 s	2 MB	High	High	$2f + 1$	O	X	O	High
Proof of Stake	Chain-based/Pure PoS[d]	100	lm	32 KB	High	Low	2/ + 1	O	X	O	High
	Chain-based/PoW/PoS Hybrid[e]	8	8 m 30 s	1 MB	High	Mid	$2f + 1$	O	X	O	High
	DPoS[f]	1000 +	0.5 s	Dynamic	High	Low	3/ + 1 $2f$ + 1 [s] >	O	O	O/x	High
	BFT PoS[g]	−14,000	(6 + 0.57?)s *R:* rounds	N/A	Mid	Low	3/ + 1 $2f$ + 1»	O	O	O/x	High
Vote-based	PBFT[h]	Mid	Low	N/A	Low	Low	3/ + 1	O	O	X	Low
	HotStuff[i]	High	Low	N/A	High	Low	3/ + 1	O	O	X	Low

a Bitcoin
b Ethereum
c DASH
d Nxt
e Peercoin
f EOS
g Tendermint
h 3f + 1 for validators of the voting process, 2f + 1 for delegate selection
i TPS, Latency was qualitatively compared to proof-based algorithms, (e.g., Diem: −1000 TPS, 10 s latency)

have malicious influence in the mining/minting process. Meanwhile, vote-based algorithms have $3f + 1$ tolerance due to their Byzantine fault-tolerant property.

Interestingly, fault (adversary) tolerance of DPoS and BFT PoS can be approached in two ways. First, if we focus on the election process, the algorithms have $2f + 1$ tolerance since an adversary with more than 51% of stake in the network could take advantage of the election. On the other hand, if we focus on the delegates' voting process, we need $3f + 1$ Byzantine fault tolerance.

As mentioned earlier (Sect. 3.2), proof-based algorithms come with an anti-Sybil attack at their core. Launching Sybil attacks involves multiple fraudulent identities, but proof-based algorithms do not depend on the number of identities; instead, it is the amount of resource/cost that influences the network operations. On the contrary, vote-based algorithms are vulnerable to Sybil attacks if there is no additional PoS or PoW admission mechanisms such as DPoS and BFT PoS [64]. This can be related to decentralization properties, where algorithms that have Sybil protection can be deployed in a permissionless environment with high resistance to censorship. In contrast, those with no Sybil protection should be permissioned, but a closed permissioned system (most likely private or consortium blockchains) may be prone to censorship.

Double-spending prevention can be related to the *finality* mechanism of the algorithm. Chain-based algorithms have *probabilistic finality*, where the immutability of the transaction increases with the number of block confirmations and can be targeted for a double-spending attack. In contrast, the algorithms that utilize the voting process have *deterministic finality*, and a transaction is finalized at the point of block creation, mitigating such attacks.

5 Conclusion

A consensus algorithm lies at the core of a blockchain and offers many research opportunities. The design of a consensus algorithm directly impacts the performance and characteristics of the developed blockchain applications. Thus, it can be argued that a thorough evaluation of a consensus algorithm is mandatory. After reviewing the literature on consensus algorithms and their evaluation criteria, we proposed an evaluation framework with comprehensive criteria for consensus algorithms in terms of performance, security, and decentralization. An in-depth evaluation framework targeted for each group of consensus algorithms (e.g., BFT variants, ASIC-resistant PoWs) is proposed as future work. We also presented the operations and characteristics of representative consensus algorithms and evaluated them with our proposed framework. We infer from our results that no consensus algorithm deems ideal for all situations. Thus, finding a well-balanced consensus algorithm that satisfies given requirements will be crucial for wide-scale adoption of blockchains across various industries and businesses.

References

1. Lam, E.: Bitcoin Hits $1 trillion value as crypto leads other assets. https://www.bloomberg.com/news/articles/2021-02-19/bitcoin-nears-1-trillion-value-as-crypto-jump-tops-other-assets (2021). Last accessed15 Mar 2021
2. Buchholz, K.: How common is crypto? https://www.statista.com/chart/18345/crypto-currency-adoption/ (2021). Last accessed 15 Mar 2021
3. PwC: Time for trust: how blockchain will transform business and the economy. https://www.pwc.com/hu/en/kiadvanyok/assets/pdf/Time_for_Trust_The%20trillion-dollar_reasons_to_rethink_blockchain.pdf (2020). Last accessed 15 Mar 2021
4. Zheng, Z., et al.: An overview of blockchain technology: architecture, consensus, and future trends. In: 2017 IEEE international congress on big data, IEEE, pp. 557–564 (2017)
5. Alsunaidi, S.J., Alhaidari, F.A.: A survey of consensus algorithms for blockchain technology. In: 2019 International Conference on Computer and Information Sciences (ICCIS), IEEE, pp. 1–6 (2019)
6. Croman, K., et al.: On scaling decentralized blockchains. In: International conference on financial cryptography and data security, pp. 106–125. Springer, Berlin, Heidelberg (2016)
7. Mingxiao, D., et al.: A review on consensus algorithm of blockchain. In: 2017 IEEE international conference on systems, man, and cybernetics (SMC), IEEE, pp. 2567–2572 (2017)
8. Nguyen, G.-T., Kim, K.: A survey about consensus algorithms used in blockchain. J. Inform. Proc. Syst. **14**(1), 101–128 (2018)
9. Hasanova, H., et al.: A survey on blockchain cybersecurity vulnerabilities and possible countermeasures. Int. J. Network Manage. **29**(2), e2060 (2019)
10. Bano, S., et al.: SoK: Consensus in the age of blockchains. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies, pp. 183–198 (2019)
11. Ferdous, M.S., et al.: Blockchain consensuses algorithms: a survey (2020). arXiv preprint arXiv:2001.07091
12. Bamakan, S.M.H., Motavali, A., Bondarti, A.B.: A survey of blockchain consensus algorithms performance evaluation criteria. Expert Syst. Appl. 113385 (2020)
13. Chaudhry, N., Yousaf, M.M.: Consensus algorithms in blockchain: Comparative analysis, challenges and opportunities. In: 2018 12th International Conference on Open Source Systems and Technologies (ICOSST), IEEE, pp. 54–63 (2018)
14. Coincheckup. https://coincheckup.com/. Accessed 15 March 2021
15. Bitinfocharts. https://bitinfocharts.com/comparison/confirmationtime-btc-ppc.html. Accessed 15 March 2021
16. Wan, S., et al.: Recent advances in consensus protocols for blockchain: a survey. Wireless Netw. **26**(8), 5579–5593 (2020)
17. Digiconomist: Bitcoin energy consumption index. https://digiconomist.net/bitcoin-energy-consumption (2021). Last Accessed 15 Mar 2021
18. Mora, C., et al.: Bitcoin emissions alone could push global warming above 2 C. Nat. Clim. Chang. **8**(11), 931–933 (2018)
19. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. In: Concurrency: the Works of Leslie Lamport, pp. 203–226 (2019)
20. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: OSDI, vol. 99, pp. 173–186. (1999)
21. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. (2008) https://bitcoin.org/bitcoin.pdf Last accessed 15 Mar 2021
22. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: International conference on financial cryptography and data security, pp. 436–454. Springer, Berlin, Heidelberg (2014)
23. Blockchaininfo: https://www.blockchain.com/charts/pools (2022). Last accessed 14 Feb 2022
24. Douceur, J.R.: The sybil attack. In: International workshop on peer-to-peer systems, pp. 251–260. Springer, Berlin, Heidelberg (2002)

25. Neudecker, T., Hartenstein, H.: Network layer aspects of permissionless blockchains. IEEE Communications Surveys & Tutorials **21**(1), 838–857 (2018)
26. Mohaisen, A., Kim, J.: The sybil attacks and defenses: a survey (2013). arXiv preprint arXiv: 1312.6349
27. Zhang, S., Lee, J.-H.: Double-spending with a sybil attack in the bitcoin decentralized network. IEEE Trans. Industr. Inf. **15**(10), 5715–5722 (2019)
28. Coinmarketcap: What is censorship resistance? https://coinmarketcap.com/alexandria/article/what-is-censorship-resistance (2020). Last accessed 15 Mar 2021
29. Back, A.: Hashcash-a denial of service counter-measure. ftp://sunsite.icm.edu.pl/site/replay.old/programs/hashcash/hashcash.pdf (2002). Last accessed 15 Mar 2021
30. Cho, H.: SIC-resistance of multi-hash proof-of-work mechanisms for blockchain consensus protocols. IEEE Access **6**, 66210–66222 (2018)
31. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum project yellow paper. https://ethereum.github.io/yellowpaper/paper.pdf (2021). Last accessed 15 Mar 2021
32. Higgins, S.: Bitmain confirms release of first ethereum ASIC miners. https://www.coindesk.com/bitmain-confirms-release-first-ever-ethereum-asic-miners. (2018) Last accessed 15 Mar 2021
33. O'Neal, S.: ETH miners will have little choice once ethereum 2.0 launches with PoS. https://cointelegraph.com/news/eth-miners-will-have-little-choice-once-ethereum-20-launches-with-pos (2020). Last accessed 15 Mar 2021
34. Percival, C.: Stronger key derivation via sequential memory-hard functions. https://www.tarsnap.com/scrypt/scrypt.pdf (2009). Last accessed 15 Mar 2021
35. Percival, C., Josefsson, S.: The scrypt password-based key derivation function, RFC 7914. https://tools.ietf.org/html/rfc7914 (2016). Last accessed 15 Mar 2021
36. Litecoin: https://litecoin.org/ (2021). Last accessed 15 Mar 2021
37. Dogecoin: https://dogecoin.com/ (2021). Last accessed 15 Mar 2021
38. Medium: What is memory-hard? https://medium.com/Linzhi/what-is-memory-hard-45a363b59dfe (2019). Last accessed 15 Mar 2021
39. DASH: X11 Hash algorithm. https://docs.dash.org/en/stable/introduction/features.html#x11-hash-algorithm (2021). Last accessed 15 Mar 2021
40. Bertoni, G., et al.: Keccak specifications. Submission to NIST (round 2), pp. 320–337. (2009)
41. Black, T., Weight, J.: X16R ASIC resistant by design. https://ravencoin.org/assets/documents/X16R-Whitepaper.pdf (2018). Last accessed 15 Mar 2021
42. Colvin, G., Lanfranchi, A., Carter, M.: EIP-1057: ProgPoW, a programmatic proof-of-work, ethereum improvement proposals, no. 1057. https://eips.ethereum.org/EIPS/eip-1057 (2018). Last accessed 15 Mar 2021
43. Medium: 'Loaded' PoW: a new direction in proof-of-work algorithms. https://jeffreyemanuel.medium.com/loaded-pow-a-new-direction-in-proof-of-work-algorithms-ae15ae2ae66a (2018). Last accessed 15 Mar 2021
44. Bitcointalk: Proof of stake instead of proof of work. https://bitcointalk.org/index.php?topic=27787.0 (2011). Last accessed 15 Mar 2021
45. King, S., Nadal, S.: PPcoin: peer-to-peer crypto-currency with proof-of-stake. https://decred.org/research/king2012.pdf (2012). Last accessed 15 Mar 2021
46. Nxt: Nxt whitepaper. https://nxtdocs.jelurida.com/Nxt_Whitepaper (2021). Last accessed 15 Mar 2021
47. PeerCoin: PeerCoin docs. https://docs.peercoin.net/ (2021). Last accessed 15 Mar 2021
48. Larimer, D.: Delegated proof-of-stake (dpos). Bitshare whitepaper **81**, 85 (2014)
49. Kwon, J.: Tendermint: Consensus without mining. Draft v. 0.6, fall 1(11) (2014)
50. Tendermint: Tendermint core. https://docs.tendermint.com/master/ (2021). Last accessed 15 Mar 2021
51. Lamport, L.: Paxos made simple. ACM Sigact News **32**(4), 18–25 (2001)
52. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: 2014 USENIX Annual Technical Conference, pp. 305–319 (2014)

53. Benet, J.: Ipfs-content addressed, versioned, p2p file system (2014). arXiv preprint arXiv: 1407.3561
54. Hyperledger Fabric: release-2.2. https://hyperledger-fabric.readthedocs.io/en/release-2.2/ (2020). Last accessed 21 Mar 2021
55. Aublin, P-L., Mokhtar, S.B., Quéma, V.: Rbft: Redundant byzantine fault tolerance. In: 2013 IEEE 33rd International Conference on Distributed Computing Systems, IEEE, pp. 297–306 (2013)
56. NeoReserach: Delegated byzantine fault tolerance: technical details, challenges and perspectives. https://github.com/NeoResearch/yellowpaper/blob/master/sections/08_dBFT.md (2019). Last accessed 21 Mar 2021
57. Bessani, A., Sousa, J., Alchieri, E.E.: State machine replication for the masses with BFT-SMART. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, IEEE, pp. 355–362 (2014)
58. Yin, M., et al.: Hotstuff: Bft consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, pp. 347–356 (2019)
59. Diem Association: Diem White Paper v2.0. https://www.diem.com/en-us/white-paper/ (2021). Last accessed 21 Mar 2021
60. Hyperledger Sawtooth: Sawtooth PBFT. https://sawtooth.hyperledger.org/docs/pbft/releases/latest/index.html (2018). Accessed 15 Mar 2021
61. InDiem Blockchain Explorer: https://indiem.info/ (2021). Accessed 21 Mar 2021
62. Amsden, Z., et al.: The libra blockchain. https://mitsloan.mit.edu/shared/ods/documents/?PublicationDocumentID=5859 (2019). Last accessed 21 Mar 2021
63. Vukolić, M.: The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In: International workshop on open problems in network security, Springer, pp. 112–125 (2015)
64. Natoli, C., et al.: Deconstructing blockchains: A comprehensive survey on consensus, membership and structure (2019). arXiv preprint arXiv:1908.08316