A Survey of Distributed Consensus Protocols for Blockchain Networks

Yang Xiao[1], Ning Zhang[2], Wenjing Lou[1], Y. Thomas Hou[1],

[1]Virginia Polytechnic Institute and State University
[2]Washington University in St. Louis

***Abstract*—Since the inception of Bitcoin, cryptocurrencies and the underlying blockchain technology have attracted an increasing interest from both academia and industry. Among various core components, consensus protocol is the defining technology behind the security and performance of blockchain. From incremental modifications of Nakamoto consensus protocol to innovative alternative consensus mechanisms, many consensus protocols have been proposed to improve the performance of the blockchain network itself or to accommodate other specific application needs.**

In this survey, we present a comprehensive review and analysis on the state-of-the-art blockchain consensus protocols. To facilitate the discussion of our analysis, we first introduce the key definitions and relevant results in the classic theory of fault tolerance which helps to lay the foundation for further discussion. We identify five core components in a blockchain, namely, block proposal, block validation, information propagation, block finalization, and incentive mechanism. Various blockchain consensus protocols are then compared and analyzed using the five-component analysis framework. These analyses provide us new insights in the fundamental differences of various proposals in terms of their suitable application scenarios (i.e. key assumptions), expected fault tolerance, scalability, and drawbacks.

We believe this survey will provide blockchain developers and researchers a comprehensive view on the state-of-the-art consensus protocols and facilitate the process of designing future protocols.

***Index Terms*—Blockchain, distributed consensus, fault tolerance, protocol design.**

I. Introduction

Since Bitcoin's inception in late 2008, cryptocurrencies and blockchain technology have piqued great interest from financial industry and the society as a whole. Blockchain, as the underlying technology behind Bitcoin, is essentially a distributed database system. The blockchain itself is a database that keeps track of all transactions occurred in the network and it is replicated at every participating full node in the network. A distributed consensus protocol running at every node controlling message exchange and local decision making ensures the consistency of the replicated databases so that all participants collectively maintain a common transaction ledger.

Blockchain is widely cited as a fully decentralized system and a secure-by-design technology. It does not require a centralized authority to bootstrap the trust among participants or to finally clear the transactions. It does not assume trust among the participating nodes. It is meant to enable trusted computation among a group of participants who do not trust each other. Blockchain is also known for providing trustworthy immutable record keeping service. The block data structure adopted in a blockchain system embeds the hash of the previous block in the next block generated. The use of hash chain ensures that data written on the blockchain can not be modified. In addition, a public blockchain system supports third-party auditing and some blockchain systems support a high level of anonymity, that is, a user can transact online using a pseudonym without revealing his/her true identity.

The security properties promised by blockchain is unprecedented and truly inspiring. Bitcoin has greatly impacted the digital payment system. It is envisioned that blockchain technology and applications built on top of it will revolutionize a broad array of financial service industries as well as many non-financial sectors. Several blockchain platforms have been designed and implemented in recent years. Among the many technical components that a blockchain system is composed of, the distributed consensus protocol is the key technology that enables blockchain's decentralization, or more specifically, that ensures all participants agree on a unified transaction ledger without the help of a central authority. The distributed consensus protocol specifies message passing and local decision making at each node. Various design choices in the consensus protocol can greatly impact a blockchain system's performance, including its capacity, scalablity, and fault tolerance.

Nakamoto consensus protocol is the protocol implemented in the Bitcoin network [1]. With the help of the Nakamoto consensus protocol, Bitcoin became the first digital currency system to resist double-spending attacks in a decentralized peer-to-peer network with little trust. As the Bitcoin network continues to grow, the proof-of-work (PoW) based Nakamoto consensus protocol has encountered several performance bottlenecks and sustainability problems. Researchers in the blockchain communities have raised the following concerns of PoW: 1) unsustainable energy consumption, 2) low transaction capacity and poor scalability, 3) long-term security concerns as mining rewards diminish. For instance, the Bitcoin network currently consists of roughly ten thousand nodes [2], while the maximum transaction capacity of Bitcoin is 7 tx/sec and can be increased up to 25 tx/sec by tuning protocol parameters without jeopardizing consensus safety [3]. In contrast, the VISA network consists of 50 million participants and can handle up to 65,000 transactions per second [4]. A single Bitcoin transaction (April 2018) consumes the equivalent electricity that can power 14.46 average U.S.

households for one day [5].

In response to the above performance limitations of PoW-based mining, blockchain researchers have been investigating new block proposing mechanisms such as proof of stake (PoS), proof of authority (PoA), and proof of elapsed time (PoET) which do not require computation-intensive mining, thus effectively reducing energy consumption. In some cases, cryptographic methods can be used to establish trust among nodes, enabling the use of more coordinated block proposing schemes such as round-robin and committee-based block generation. Appropriate incentive that will continue to encourage honest participation in the blockchain network is another key component of consensus protocol. Therefore, these alternative block proposing schemes are often accompanied by a new block reward scheme that promotes participation fairness and increases overall system sustainability. Popular blockchain consensus protocols encompassing these ideas include Peercoin [6], Bitcoin-NG [7], Ourosboros (Cardano) [8], Snow White [9], and EOS.IO [10], POA Network [11], etc.

Besides block proposing and incentive mechanisms, researchers have been seeking solutions from the prevailing wisdom—primarily classical BFT consensus and secure multiparty computation—for more efficient block finalization methods. For example, the state machine replication (SMR) based BFT consensus algorithms have great potential in permissioned blockchain networks operated with static and revealed identities, of which Tendermint [12], Algorand [13], Casper FFG [14], and Hyperledger Fabric [15] are well-known use cases. Moreover, asynchronous consensus protocols such as HoneyBadgerBFT [16] and BEAT [17] were proposed to provide robust block finalization under severe network conditions with uncertain message delays.

Our contribution With more blockchain consensus mechanisms being proposed, there is a pressing need to analyze and compare them in a formal and cohesive manner. In this survey we present a comprehensive review and analysis of the state-of-the-art blockchain consensus protocols and their development history, with a special focus on their performance, fault tolerance, and security implications. Our information sources include academic papers, consensus protocol white papers, official documentation and statistics websites of cryptocurrencies. Specifically, our survey features the following contributions:

1) providing a succinct background of classical distributed consensus research, including both partially synchronous and asynchronous BFT protocols applicable to blockchain consensus;
2) analyzing a broad array of blockchain consensus protocols in a proposed five-component framework which consists of block proposal, block validation, information propagation, block finalization, and incentive mechanism; and identifying available options for each component;
3) providing abstractions for four classes of proof of stake (PoS) based consensus protocols and a review of their real-world examples;
4) providing a qualitative comparison of all mentioned consensus protocols with respect to the five-component framework, fault tolerance, and potential transaction capacity.

Previous surveys The comparison study by Vukolic [18] treats two major genres of blockchain consensus protocols, namely PoW-based and BFT-based, with respect to transaction capacity, scalability limits, and security implications. PoW-based protocols scale well with network size and allow for permissionless access, but they support a very limited transaction capacity. While BFT protocols achieve much higher transaction capacity, they need a permissioned network for identity management and do not scale well with network size. Besides remarking their differences, this paper also suggests the hybrid use of BFT and PoW to enhance the performance of established blockchains.

Cachin et al. [19] gives an overview of thirteen prominent consensus protocols designed for permissioned blockchain platforms along with their fault tolerance and security properties. The authors also made a powerful statement that the design of blockchain consensus protocols should follow the rigor established in prevailing wisdom of cryptography, security, and distributed systems, rather than in an ad hoc manner. This is particularly true when the consensus protocol regulates significant financial values and societal trust. However, it does not provide a systematic method to combine the prevailing wisdom in order to design a consensus protocol for specific needs.

The work by Bano et al. [20] is the first well-structured survey of blockchain consensus protocols. It identifies three classes of consensus protocols based on committee formation and block proposing rules: 1) PoW, 2) proof of X alternatives to PoW, and 3) hybrid protocols that take advantage of classical distributed consensus techniques. Specially this paper presents an evaluation framework that takes into account the protocol safety (censorship resistance, DoS resistance and fault tolerance) and performance (throughput and latency). Notably, this paper is the closest work to our survey in terms of classification of block proposing rules. However, analysis of new protocols in the area of PoS require new methods of analysis.

Wang et al. [21] provides a comprehensive survey of blockchain consensus protocols, with particular interests on the cryptographic methods that can be used to enhance protocol security. Specially, the survey starts with a framework based on zero-knowledge proof (ZKP) and extends to three aspects of blockchain consensus: 1) Proof-of-X consensus protocols for permissionless blockchain, 2) incentive mechanisms in the framework of Nakamoto PoW and node strategies, 3) virtual mining schemes such as proof of stake (PoS) and their hybridization with classical BFT protocols. This paper also explores the efficiency-scalability tradeoff and the cost of decentralization. Although great details are presented, this survey does not provide a cohesive analysis framework that captures different functional components of a consensus protocol.

Xiao et al. [22] provides a succinct tutorial on distributed consensus protocols, from classical BFT protocols to the Nakamoto consensus protocol as well as recent breakthroughs in blockchain consensus. Specially this paper provides an abstraction of consensus goals for permissionless blockchains following the paradigm of classical BFT consensus. However, this tutorial does not clarify how different components of a blockchain consensus protocol may contribute to the performance and security of the protocol differently.

The remaining part of this survey is organized as follows. Section II provides a succinct summary of classical fault tolerant consensus in distributed systems. Several legacy BFT consensus protocols under both partially synchronous and asynchronous network assumption are also presented. Section III presents the background knowledge of blockchain and the consensus goals, and introduces the five essential components of a blockchain consensus protocol. Section IV focuses on the well-known Nakamoto consensus protocol—the defining technology of Bitcoin, and its most infamous vulnerabilities. Section V provides a systematic overview of the proof-of-stake (PoS) protocols, the most promising competitors to the PoW-based Nakamoto consensus. Section VI discusses alternative consensus protocols that can be used under specific application scenarios. Section VII qualitatively compares all blockchain consensus protocols studied and summarizes their design philosophy. Section VIII concludes the paper.

II. Fault-Tolerant Distributed Consensus

The fault-tolerant distributed consensus problem has been extensively studied in distributed systems since the late 1970s and recently gained popularity in the blockchain community, especially for permissioned blockchains where every consensus participant reveals its identity. Generally, consensus in a distributed system represents a state that all participants agree on the same data values. Depending on the medium for message exchange, distributed systems are often classified into two types: *message passing* and *shared memory* [23]. In this section we are interested in message passing systems because of their resemblance to contemporary blockchain systems, wherein distributed consensus on a single network history is reached through peer-to-peer communication. We will use the terms process/node/server interchangeably, as they all refer to an individual participant of distributed consensus.

A. System Model

1) Distributed system and task: We consider a distributed system that consists of N independent processes. Each process p_i begins with an individual initial value x_i and communicates with others to update this value. Each local value can be used for a certain task, such as computation or just storage. If the processes are required to perform the same task, consensus on a single value is required before they proceed to the task.

2) Process failure: A process suffers a *crash failure* if it abruptly stops working without resuming. The common causes of a crash failure include power shutdown, software errors, and DoS attacks. A *Byzantine failure*, however, is much severer in that the process can act arbitrarily while appearing normal. It can send contradicting messages to other processes in hope of sabotaging the consensus. "Byzantine" was coined by Lamport et al. [24] in 1982 when describing *Byzantine Generals Problem*, an allegory for single-value consensus among distributed processes. The common cause of a Byzantine failure is adversarial influence, such as malware injection and physical device capture. And multiple Byzantine processes may collude to deal more damage.

3) Network synchrony: Network synchrony defines the level of coordination among all processes. Three levels of synchrony, namely *synchronous*, *partially synchronous*, and *asynchronous*, are often assumed in the literature [22], [25].

- In a *synchronous* network, operations of processes are coordinated in rounds with clear time constraints. In each round, all processes perform the same type of operations. This can be achieved by a centralized clock synchronization service and good network connectivity. Practically, a network is generally considered synchronous if message delivery is guaranteed within a fixed delay Δ, of which the case is also called Δ-synchronous.
- In a *partially synchronous* network, operations of processes are loosely coordinated in a way that message delivery is guaranteed but with uncertain amount of delays. Within the scope of partial synchrony, *weak synchrony* requires message delay not grow faster than the elapsing time indefinitely [26], while *eventual synchrony* ensures Δ-synchrony only after some unknown instant [27]. In either case, the design of the networked processes can still follow that of a synchronous network if the time horizon is long enough.
- In an *asynchronous* network, operations of processes are hardly coordinated. There is no delay guarantee on a message except for its eventual delivery. And the coordination of processes (if there is any) is solely driven by the message delivery events. This is often caused by the absence of clock synchronization (thus no notion of shared time) or the dominance of a mighty adversary over all communication channels.

It has been shown by Fischer, Lynch, and Paterson [28] that under the asynchronous case, consensus cannot be guaranteed with even a single crash failure. This is commonly known as the FLP impossibility, named after the authors' initials. However, this impossibility can be practically circumvented using randomized decision making and a relaxed termination property, as we will discuss in II-E.

B. Byzantine Fault Tolerant Consensus

We call a consensus protocol *crash fault tolerant* (CFT) or *Byzantine fault tolerant* (BFT) if it can tolerate one or more crash or Byzantine failures while keeping normal functioning. Because of the inclusive relationship between a crash failure and Byzantine failure, a BFT consensus protocol is naturally CFT. BFT consensus is defined by the following four requirements [22], [25], [27]:

- *Termination:* Every non-faulty process decides an output.
- *Agreement:* Every non-faulty process eventually decides the same output $\hat{y}$.
- *Validity:* If every process begins with the same input $\hat{x}$, then $\hat{y} = \hat{x}$.
- *Integrity:* Every non-faulty process' decision and the consensus value $\hat{y}$ must have been proposed by some non-faulty process.

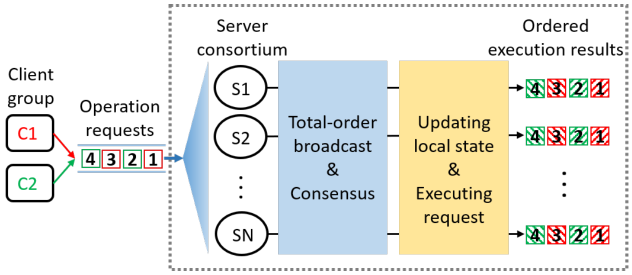


Fig. 1. A high-level illustration of SMR-based distributed computing that serves four operation requests from two clients.

These four requirements provide a general initial target for distributed consensus protocols. For any consensus protocol to attain these BFT consensus conditions, the underlying distributed network should satisfy the following condition: $N \geq 3f+1$ where f is the number of Byzantine processes. This fundamental result was first proved by Pease et al. [29] in 1980 and later adapted to the BFT consensus framework. The proof involves induction from $N=3$ and partitioning all processes into three equal-sized groups, with one containing the faulty ones. Interested readers are referred to [29], [25] for detailed proofs.

C. Consensus in Distributed Computing

Consensus in distributed computing is a more sophisticated realization of the aforementioned distributed system. In a typical distributed computing framework, one or more clients issue operation requests to the server consortium, which provide timely and correct computing service in response to the requests despite some of servers may fail. Here the correctness requirement is two-fold: correct execution results for all requests and correct ordering of them. According to Alpern and Schneider's work on liveness definition [30] in 1985, the correctness of consensus can be formulated by two requirements: *safety*—every server correctly executes the same sequence of requests, and *liveness*—all requests should be served.

To fulfill these requirements even in the presence of faulty servers, server replication schemes especially *state machine replication* (SMR) are often heralded as the de facto solution. SMR, originated from Lamport's early works on clock synchronization in distributed systems [31], [32], was formally presented by Schneider [33] in 1990. Setting in the client-server paradigm, the core idea of SMR can be summarized as follows:

1) All servers start with the same initial state;
2) All servers receive the same sequence of requests as how they were generated from clients (total-order broadcast);
3) All servers receiving the same request shall output the same execution result and end up in the same state.

To be noted, total-order broadcast is also known as *atomic broadcast* (ABC) [34], which is in contrast to the *reliable broadcast* (RBC) primitive [35]. The latter only requires all servers receive the same requests without considering order. It is shown in [36], [37] that atomic broadcast and distributed consensus are equivalent problems.

A high-level diagram of SMR-based distributed computing is illustrated in Figure 1. The N-server consortium accepts client requests and confirms each other's state before reaching consensus and executing requests. In many cases, especially randomized consensus protocols, there can be an alternating procedure of total-order broadcasts and local state updates until a certain consensus target is met, before moving on to execution. Practically, SMR is often implemented in a leader-based fashion. A primary server (say S1 in Figure 1) receives client requests and starts the broadcast procedure so that the other $N-1$ replica servers receive the same requests and update the local states to that of the primary.

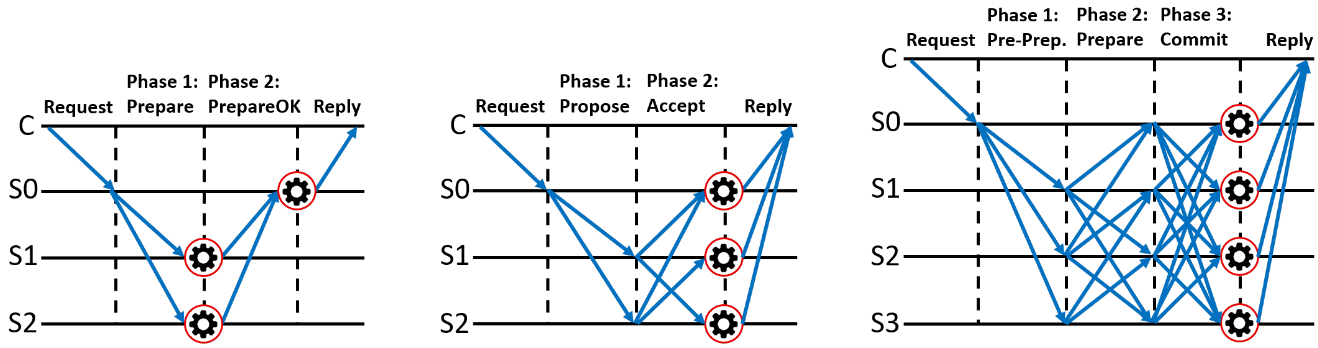
In the rest of this section we will summarize several well-known consensus protocols (some are based on SMR) under different network synchrony assumptions.

D. Consensus Protocols for Partially Synchronous Network

The ground-breaking work by Dwork, Lynch, and Stockmeyer [27] in 1988 laid the theoretical foundation of partially synchronous consensus. By dissecting the consensus objective into *termination* and *safey*, the authors are able to formally prove the feasibility of four consensus goals, including CFT, omission-tolerance, BFT, unauthenticated BFT, under the Δ-synchrony/eventual synchrony condition. Notably, this work jump-started research endeavors on consensus in partially synchronous systems, including PBFT which we will discuss in this section.

1) DLS protocol: The same paper [27] we just discussed also proposes a prototype consensus protocol (called DLS for authors' namesake) featuring a broadcast primitive for each consensus cycle. Specifically, the broadcast primitive is started by an arbitrary process p and consists of two initial rounds and subsequent iterative rounds. Through message exchanges in each round, the iterative procedure eventually drives the processes to reach agreement on a common value (either the one proposed by p or a default value). At message complexity $O(N^2)$ (N is the number of processes), the broadcast primitive essentially enables the DLS protocol to tolerate f Byzantine processes if the population $N \geq 3f+1$. The cryptocurrency Tendermint uses an adapted version of DLS for block finalization.

2) Viewstamped Replication (VR): Proposed by Oki and Liskov [38] in 1988, viewstamped replication is a server replication scheme that handles server crashes. It was later extended into a consensus protocol by Liskov and Cowling [39] in 2012, which we will refer to as VR. VR is a SMR scheme designed in the client-server fashion and consists of three sub-protocols: 1) normal operation, 2) view change, 3) recovery. The primary server receives a client request and starts the normal operation of consensus, as is shown in Figure 2(a). In the case of a crash failure of the primary, the view change protocol is triggered at every replica per the timeout of the prepare message. They broadcast view-change messages to each other and count the receptions. After receiving view-change messages from more than half of the replicas, the next-in-line replica becomes the new primary and informs the others to resume the normal operation. The



(a) VR (b) Paxos (c) PBFT

Fig. 2. Messaging diagram during the normal operation of three SMR protocols. C is the client. S0 is the primary server (leader) who receives request from the client and starts the consensus. S1+ are replica servers. Every server updates local state after receiving a message. Circled gear icon represents request execution. VR and Paxos can tolerate one crash failure when $N = 3$. PBFT can tolerate one Byzantine failure when $N = 4$.

recovery protocol is used by any server to recover from a crash. VR can tolerate f crashed replicas if the network population $N \geq 2f+1$. However it does not tolerate any Byzantine failure, because the replicas simply follow the instructions from the primary without mutual confirmation nor communicating with the client. On the flip side, this makes VR efficient, with message complexity of $O(N)$.

3) Paxos: Paxos is a pioneering SMR scheme proposed by Lamport [40] in an allegorical fashion in 1989 and later explained in 2001 [41]. It was designed specifically for fault tolerant consensus while bearing many similarities to VR. Paxos classifies nodes into three roles: proposers, acceptors, and learners. A proposer suggests a value in the beginning and the goal is to make acceptors agree on a single value, and learners learn this value from acceptors. In the client-server scenario depicted in Figure 2(b), the client is the learner, the primary is the proposer, and the replicas are acceptors. After updating to the same state, all servers execute the request and send it to the client who then chooses the majority result. When the proposer suffers a crash failure, the acceptors elects a new leader through a similar propose-accept procedure. Akin to VR, Paxos can tolerate f crashed acceptors when $N \geq 2f+1$, but no Byzantine failures. Because of the mutual messaging during the accept phase, the message complexity of Paxos is $O(N^2)$.

Embarking from its original design, Paxos has grown into a family of consensus protocols of which each has a specific feature, including multi-Paxos, cheap-Paxos, and fast-Paxos. Raft, a new SMR consensus protocol developed by Ongaro and Ousterhout [42] in 2014, is based off Paxos but with a more understandable design logic.

4) Practical Byzantine Fault Tolerance (PBFT): Developed by Castro and Liskov [26] in 1999, PBFT is the first SMR-based BFT consensus protocol that has gained wide recognition for practicality. It has become almost a synonym of BFT consensus protocols in the blockchain community. PBFT was designed from the framework of VR and also took inspiration from Paxos. PBFT consists of three sub-protocols: 1) normal operation, 2) checkpoint, 3) view change. The normal operation protocol is shown in Algorithm 1 and Figure 2(c).

Algorithm 1: PBFT (normal operation protocol)

/* Request */
Client sends an operation request to the primary;
/* Phase 1: Pre-prepare */
The primary relays this request to replicas via *pre-prepare* messages;
Replicas record the request and update local states;
/* Phase 2: Prepare */
Replicas send *prepare* messages to all servers (replicas and the primary);
Once receiving $\geq 2f+1$ *prepare* messages, a server updates local state and is ready to commit;
/* Phase 3: Commit */
Servers send *commit* messages to each other;
Once receiving $\geq 2f+1$ *commit* messages, a server starts to execute the client request and then update local state;
/* Reply */
Every server replies its result to the client.

Ideally, all results replied to the client should be the same; otherwise the client chooses the majority result. Because of mutual state confirmation among servers in the prepare and commit phase, the consensus process guarantees the majority of operation execution results received by the client are correct as long as more than two thirds of servers (including primary) are honest. Therefore PBFT can tolerate f Byzantine replicas when the server population $N \geq 3f+1$, which is the fundamental BFT bound proved in [27]. The message complexity of PBFT normal operation is $O(N^2)$.

The checkpoint protocol serves as a logging tool that keeps a sliding window (of which the lower bound is the stable checkpoint) to track active operation requests. The stable checkpoint is used for safely discarding older requests in the operation log and facilitating the view change protocol. In the case of a primary failure, the view change protocol is triggered at every replica that detected the timeout of the primary messages. They oust the incumbent primary and broadcast view-change messages to each other and count receptions. After receiving view-change messages from $2f$ peers, the next-in-line replica becomes the new primary and informs the others to resume the normal operation.

PBFT has inspired a stock of BFT consensus protocols with enhanced security and performance. Well known proposals include Quorum/Update (QU) [43], Hybrid Quorum (HQ) [44], Zyzzyva (using speculative execution) [45], FaB [46], Spinning [47], Robust BFT SMR [48], and Aliph [49]. Interested readers are referred to Bessoni's tutorial [50] for an overview on these protocols.

E. Consensus Protocols for Asynchronous Network

For many practical distributed systems that are predominantly built upon wired communication and transport layer protocols, partial synchrony is a practical assumption. However in scenarios such as mobile ad hoc network (MANET), and delay tolerant networks (DTN), the network can be considered of near-to-none synchrony. As having been proved in the FLP impossibility result [28], consensus can not be guaranteed in a fully asynchronous network with even one crash failure. Moreover, the unreliable communication links have an equivalent effect of a Byzantine scheduler. However, this impossibility result can be practically circumvented by two primitives: *probabilistic termination* and *randomization*.

First of all, according to [51] the termination property presented in Section II can be subdivided into two classes:

- *Deterministic termination:* Every non-faulty process decides an output by round r, a predetermined parameter.
- *Probabilistic termination:* The probability that a non-faulty process is undecided after r rounds approaches zero as r grows to infinity.

For synchronous or partially synchronous networks where message delay and round period are bounded, protocols such as PBFT can exploit a timeout mechanism to detect anomaly of the primary, which makes deterministic termination an achievable goal. For asynchronous networks where messages are eventually delivered but with no timing guarantee, the consensus process can only be driven by the message delivery events. This limitation demands probabilistic termination.

To realize probabilistic termination, randomization (simultaneously proposed by Ben-Or [52] and Rabin [53] in 1983) can be instantiated in the consensus protocol. The basic idea is that a process makes a random choice when there are not enough trusted messages received for making a final decision.

Next we introduce four primitives/protocols that aim to solve asynchronous BFT consensus. Although in different contexts, they all feature probabilistic termination and make use of randomization, and some are built upon the others' ideas.

1) Bracha's RBC and asynchronous consensus protocol: Bracha et al. [54] proposed the pioneering reliable broadcast (RBC) primitive and an asynchronous consensus protocol in 1984 to solve the asynchronous Byzantine Generals Problem [24] that all non-faulty processes should eventually make the same binary decision, 1 or 0.

Bracha's RBC guarantees that non-faulty processes will never accept contradicting messages from any process and forces the faulty ones to output either nothing (mimicking the crash failure) or the correct value. Bracha's asynchronous consensus protocol (adapted from Ben-Or's 1983 protocol [52]) runs by phases and each phase contains three RBC rounds for inter-process value exchange. Here we show the round-3 of each phase, which contains the randomized step. After receiving at least $N-f$ value messages (N is the number of processes and f is an estimate count of Byzantine processes), a process P_i does the following:

1. If receiving a value v from more than $2f$ peers, decide v;
2. Else if receiving a value v from more than f peers, hold v as proposal value and go to the next phase;
3. Else, toss a coin (1/2 chance for 0 or 1) for the proposal value and go to the next phase.

When enough phases pass, the executions of step 2 and step 3 of RBC round-3 at all non-faulty processes will gradually filter out the influence of contradicting messages and eventually make the correct decision via step 1. Note this convergence only happens if the Byzantine population $f < N/3$, which is the fundamental bound of BFT consensus.

In terms of performance, the message complexity of RBC is $O(N^2)$ in each round and the expected number of rounds to reach consensus is $O(2^N)$ if $f = O(N)$, which gives a total message complexity of $O(N^2 2^N)$. If $f = O(\sqrt{N})$ (the benign case), it is shown in [52], [54] that the expected number of rounds to reach consensus for the randomized protocol is a constant, yielding a total message complexity of $O(N^2)$.

2) Ben-Or's ACS protocol for MPC: Agreement on a common subset (ACS) was used by Ben-Or et al. [55] in 1994 as a consensus primitive for secure and efficient multi-party computation (MPC) under asynchronous setting. In a network of N players, each player holds a private input x_i that was acquired secretly. The goal of MPC is to let the players collectively compute a function $\mathscr{F}(x_1, ..., x_N)$ and obtain the same result. Assuming f players can be faulty, the ACS primitive requires the players to agree on a subset *ComSubset* of at least $N-f$ honest inputs, which are then used for computing $\mathscr{F}(\cdot)$.

Ben-Or's ACS protocol builds on two primitives: RBC and binary asynchronous Byzantine agreement (ABA) which allows players to agree on the value of a single bit. Bracha's RBC [54] and Canetti et al.'s Fast ABA [56] are suggested respectively in [55] and used as black-boxes. Algorithm 2 shows the ACS protocol at each player. In the end, there will be at least $N-f$ completed ABA instances with output 1, yielding a $\mathscr{F}$-computable *ComSubset*.

Because both Bracha's RBC and Canetti's ABA can tolerate f Byzantine players when $N \geq 3f+1$, the same fault tolerance result is inherited by Ben-Or's ACS protocol. For complexity analysis, Bracha's RBC (in the benign case) and Canetti's ABA have message complexity of $O(N^2)$ and $O(N^3)$ [56] respectively, and all ABA instances end in constant rounds. As a result, Ben-Or's ACS protocol has an overall communication complexity (in bits) of $O(mN^2 + N^3)$ at each player, where m is the maximum bit-size of any input.

Algorithm 2: Ben-Or's ACS protocol (at player P_i)

```
/* Phase 1: Reliable Broadcast                    */
Start RBC_i to propose my input x_i to the network;
Participate in other RBC instances;
/* Phase 2: Asynchronous BA                       */
while round ≤ MaxRound do
    if receiving x_j from RBC_j then
        Join ABA_j with input 1;
    end
    if completion of N − f ABA instances then
        Join other BA instances with input 0;
    end
    if completion of all N ABA instances then
        ComSubset = {x_k | ABA_k outputs 1};
        return ComSubset;
    end
end
```

As we will see next, ACS can be conveniently adapted to asynchronous BFT consensus for blockchain systems, by substituting inputs with transaction sets.

3) HoneyBadgerBFT: Proposed by Miller et al. [16] in 2016, HoneyBadgerBFT is the first asynchronous BFT consensus protocol specifically designed for blockchain. It essentially realizes atomic broadcast: N players with different sets of transactions work to agree on a common set of sorted transactions that will be included in a block.

Although taking the multi-value Byzantine agreement (MVBA) primitive by Cachin et al. [57] as the benchmark, HoneyBadgerBFT actually follows Ben-Or's ACS construction [55] for better communication efficiency. HoneyBadgerBFT's ACS cherry-picks the design of its sub-components: Cachin and Tessaro's erasure-coded RBC [58] and Mostéfaoui et al.'s common-coin based ABA [59]. They together incur $O(mN + N^2 \log N)$ communication complexity at each player. To prevent an adversary from censoring particular transactions, threshold public key encryption (TPKE) [60] is used before ACS so that consensus is performed on ciphertexts. In the decryption phase when a player receives enough shares from peers (generated by *TPKE.DecShare*) that exceed a threshold quantity, it then proceeds to the actual decryption task (*TPKE.Dec*) and sorts the transactions. The communication complexity of the decryption process is $O(N^2)$. Figure 3 illustrates the workflow of these components in one block cycle.

HoneyBadgerBFT processes transactions in batches. Let B be the predefined batch size, denoting the maximum number of transactions that a block may enclose. For every block cycle, each player proposes a set of B/N transactions, which are randomly chosen from recorded transactions. This is to ensure transaction sets proposed by different players are mostly disjoint so as to maximize throughput. Assuming the average bit-size of a transaction set $m = \frac{|t|B}{N} >> N$ where $|t|$ is the average transaction bit-size. Then the protocol's communication overhead will be dominated by the RBC, yielding overall communication complexity of $O(|t|B)$ at one player, or $O(|t|N)$ for one transaction.

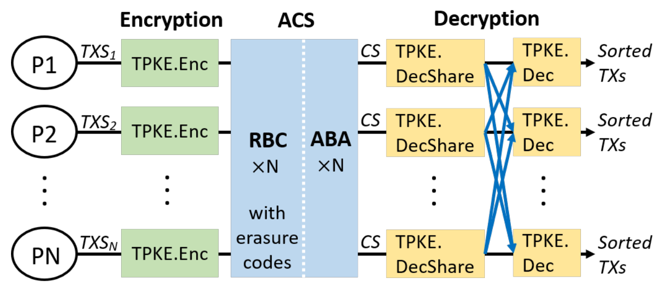


Fig. 3. HoneyBadgerBFT workflow. TXS_i is the set of transactions proposed by player P_i. CS is the common subset output of ACS, comprising of at least $N - f$ encrypted transactions. The decryption process outputs sorted transactions that will be finalized in the block.

Compared to popular partially synchronous consensus protocols such as PBFT, HoneyBadgerBFT has a higher cryptography overhead but features two advantages. First, as an asynchronous protocol HoneyBadgerBFT does not rely on a timeout mechanism for detecting the malfunctioning. This make HoneyBadgerBFT less sensitive to unpredictable network delays that might stall consensus. Second, HoneyBadgerBFT does not need a leader rotation scheme. In PBFT every round of consensus is started by a leader (the primary), while in HoneyBadgerBFT every node starts its own broadcast and Byzantine agreement instance for proposed transactions; the concurrent execution of these instances effectively saves the need of a leader. As a result, the bandwidth of any individual leader will not become the bottleneck of overall network's capacity. Currently the blockchain initiative POA Network [11] is considering to adopt HoneyBadgerBFT.

On the other hand, due to HoneyBadgerBFT's asynchronous design philosophy that consensus progress is driven by message deliveries, transaction confirmation latency is externally influenced and uncontrollable. However in various applications such as industrial control and supply chain management, low latency is often times a more important metric than throughput. As a response, Duan et al. [17] proposed BEAT in 2018, a collection of five asynchronous BFT protocols based off HoneyBadgerBFT but with different performance emphases on latency, throughput, and network scalability. Specially, features of the five constituent protocols can be mixed to achieve other objective trade-offs.

F. Blockchain Compatibility of Classical BFT Protocols

In a blockchain network, every consensus participant can validate transactions and propose new blocks. For SMR-based BFT consensus protocols that rely on a dedicated primary server to receive client requests and start the consensus, the following adaptation is needed: allowing all servers to act as a primary to propose transactions/blocks and reaching consensus on the finality of multiple transactions/blocks concurrently. For example, Tendermint [12] is based on a modified DLS protocol and allows every eligible participant to propose a block in a round-robin fashion during a block cycle. Out of all the proposed blocks the modified DLS protocol is able to finalize only one block at the end of the block cycle.

BFT protocols are also notorious for their limited scalability in network size. Epitomized by PBFT, the message complexity of partially synchronous BFT protocols grows quadratically with the network size N. This means that given a fixed network bandwidth at each node, a growing network size leads to an exploding communication overhead and a diminishing transaction capacity. According to the performance evaluation in [16], PBFT achieves a maximum per-second transaction throughput of 16,000 when $N = 8$; this figure drops to around 3,000 when $N = 64$. On the other hand, for asynchronous protocols like HoneyBadgerBFT where erasure coding and threshold encryption are used to reduce communication complexity and enhance security, the extensive use of cryptography also brings non-negligible computation overhead, adding to processing delays.

Other blockchain compatibility challenges for BFT protocols include: 1) allowing nodes to join and leave while continuing the consensus process and countering Sybil attacks; 2) adapting to real-world peer-to-peer networks that are sparsely connected. In later sections we will revisit these issues when discussing blockchain protocols that incorporate BFT consensus. On the bright side, a typical BFT protocol achieves *deterministic finality*, which is also known as *forward security* [61] in that a settled transaction will never be altered. As we will discuss in Section IV, this allows BFT protocols to take advantage of shorter block intervals and attain high throughput.

III. An Overview of Blockchain Consensus

Compared to traditional distributed computing with a clear client-server model, a blockhchain network allows every participant to be both a client (to issue transactions) and a server (to validate and finalize transactions). The underlying ledger data structure—the blockchain—is the consensus target that consists of chronologically ordered and hash-chained blocks. Each block contains a bundle of valid transactions and transactions across the blockchain should be consistent with each other (i.e. no double-/over-spending nor appropriation). Meanwhile, a blockchain system is often associated with a financial application and bears the responsibility of transaction processing and clearing. As a result, the responsibility of a blockchain consensus protocol is further-reaching than traditional distributed consensus protocols. In this section we provide a background of the blockchain network and data structure, introduce the blockchain consensus goal adapted from BFT consensus paradigm and the five-component framework that we use to analyze the blockchain consensus protocol.

A. Blockchain Infrastructure

Network The physical form of a blockchain system, as is adopted by most public cryptocurrencies, is a peer-to-peer overlay network on top of the Internet. The network formation and node bootstrapping are determined by a wire protocol based in the application layer. Specifically, a public (permissionless) blockchain allows for free join and leave without any authentication, as long as the node holds a valid pseudonym (account address) and is able to send transactions and make local decisions. On the other hand, in a private (permissioned) blockchain participants need to be authenticated before sending transactions and participating in group consensus. This security predicate of permissioned blockchain allows for the deployment of a more efficient consensus protocol [22].

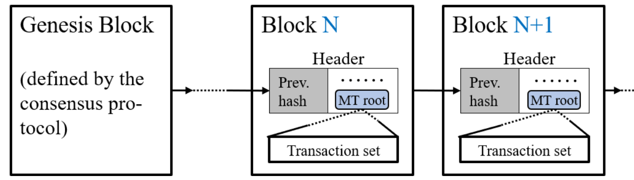


Fig. 4. Blockchain data structure. Blocks are sequentially chained together via hash pointers. The Merkle tree root (MT root) is a digest of all transactions included in a block.

Beneath the application layer of blockchain lies the basic infrastructure of the Internet. Thanks to the transport layer protocols (especially the retransmission mechanism), message delivery is considered guaranteed, while the message delay may vary but most likely won't grow longer as time elapses (weak synchrony) or remains within a certain bound (Δ-synchrony). Therefore we often consider a practical blockchain network partially synchronous, just like most distributed networks overlaying on the Internet. This allows the consensus protocol to take advantage of the timing services of the Internet. For example, in Bitcoin the partial synchrony assumption is echoed by its usage of local timestamps for loose chronological ordering, showing time consciousness. For the blockchain networks that reside on a ad hoc infrastructure not based on the Internet, the message transmission is subject to unexpected network delays, which gives rise to asynchronous consensus protocols such as HoneyBadgerBFT, as we discussed in Section II.

Transaction A blockchain transaction can be regarded as a public static data record showing the token value redistribution between senders and receivers [21]. Take Bitcoin as an example, a transaction transfers token ownership from the sender account to the receiver account(s). It specifies a list of inputs and a list of outputs, with each input claiming a previous unspent transaction output (UXTO) that belongs to the sender, who needs to attach its signature to the inputs to justify the claim. Each output specifies how many tokens go to which receiver and the total token value of the outputs is equal to the UXTOs claimed by the inputs. Therefore, we can always recover the ownership records of any specific token by tracing back the signatures along the chain of transactions. The token balance of an account equals to the summed UXTOs that belong to the account.

Blockchain data structure The blockchain is the underlying data structure for transaction ledger keeping. It is also the consensus target of the network. The basic structure of blockchain is illustrated in Figure 4. Every block records a set of transactions that should be valid and clear of double spending. As was pioneered by Bitcoin, the transactions are often organized in a Merkle tree and represented by the Merkle tree root in the block header. The block header also contains a hash of the previous block (except for the genesis block) and other configuration information, which typically includes the

TABLE I
FIVE COMPONENTS OF A BLOCKCHAIN CONSENSUS PROTOCOL.

Component	**Purpose**	**Resemblance in traditional SMR consensus protocols**	**Available options**
Block proposal	Generating blocks and attaching essential generation proofs (for Sybil attack resistance).	Combination of clients issuing operation requests the primary server starting the consensus.	Proof of work (PoW), proof of stake (PoS), proof of authority (PoA), proof of retrievability (PoR), proof of elapsed time (PoET), round robin, committee-based, etc.
Information propagation	Disseminating blocks and transactions across the network.	Reliable broadcast of operation requests.	Advertisement-based gossiping, block header soliciting, unsolicited block push (broadcast), relay network (for mining pools), etc.
Block validation	Checking blocks for generation proofs and the transactions within.	Validity check and execution of operation requests.	Proof checking (for proof-of-X block proposal), digital signature & eligibility checking (for committee-based block proposal), etc.
Block finalization	Reaching consensus on certain blocks.	Servers reaching an agreement on the execution result that will be accepted by the client.	Longest-chain rule, GHOST rule, PBFT, other Byzantine agreements, checkpointing, etc.
Incentive mechanism	Encouraging honest participation and driving the system moving forward.	N/A.	Block rewards, transaction fees, eligibility for issuing new transactions, etc.

blockchain status at block generation.

Aside from recording the transaction history, the blockchain can also record auxiliary information that can be used for other purposes. The locking and unlocking scripts associated with transaction inputs and outputs can be used to construct global state machine that is useful in many applications, such as smart contracts. And the block header may contain more fields that facilitate system coordination. For example, Ethereum's proof-of-stake (PoS) scheme Casper FFG [62] utilizes smart contract to implement the staking process; Algorand [13] attaches a cryptographic proof to each new block to show the block proposer's eligibility to propose. As a result, the blockchain can hold the necessary control information usable by the consensus protocol. We will discuss these protocols in later sections.

B. Consensus Goal

The goal of a blockchain consensus protocol is to ensure that all participating nodes agree on a common network transaction history, which are serialized in a blockchain. Based on the previous discussion on BFT consensus and the consensus goal abstraction provided in [22], we similarly define the following requirements for blockchain consensus:

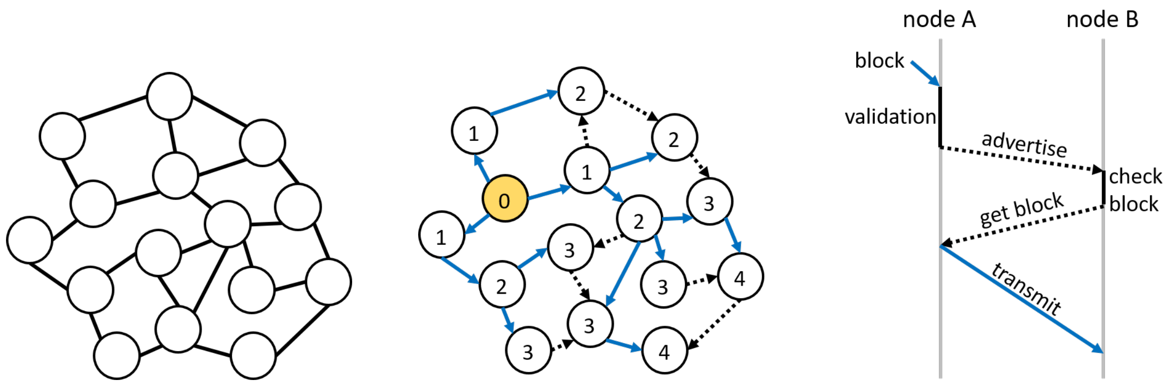
- **Termination** At every honest node, a new transaction is either discarded or accepted into the blockchain, within the content of a block.
- **Agreement** Every new transaction and its holding block should be either accepted or discarded by all honest nodes. An accepted block should be assigned the same sequence number by every honest node.
- **Validity** If every node receives a same valid transaction/block, this transaction/block should be accepted into the blockchain.
- **Integrity** At every honest node, all accepted transactions should be consistent with each other (no double spending). All accepted blocks should be correctly generated and hash-chained in chronological order.

The *termination* and *validity* requirements are similar to their counterparts in classical distributed consensus, as they indicate the system's liveness. The *agreement* requirement is enhanced with the total ordering requirement from distributed computing, underscoring the serialization of blocks and transactions. The *integrity* requirement dictates the correctness of the origin of transactions and blocks, essentially fulfilling the promise of anti-double-spending and tamper-proofing. These requirements can serve as the design principles of new blockchain initiatives. To fit into specific application scenarios, they can be tailored or supplemented with more specification.

C. Components of a Blockchain Consensus Protocol

Based on the above discussions, we identify five key components of a blockchain consensus protocol: block proposal, block validation, information propagation, block finalization, and incentive mechanism. For each component we also specify the purpose, resemblance in traditional SMR consensus protocols, and a list of available options in Table I. The available options are non-exhaustive, as many more are being developed at the time of writing.

Though the five components are all vital to successful blockchain consensus, a new blockchain consensus protocol proposal may not cover all of them. For example, the incentive mechanism is indispensable to public blockchain networks especially those carrying a financial responsibility; however in private blockchains where participation is sanctioned by the network, it is not a must-have. Interestingly, many new public blockchain initiatives have been fixating only on block proposal, while inheriting the other four components from the Nakamoto consensus protocol of Bitcoin. This is likely due to that Bitcoin's proof-of-work block proposing mechanism attracts the most criticism because of its scalability limitation and energy inefficiency. In the remaining sections we will classify consensus protocols largely by the block proposing mechanism, and provide details about other components when it comes to specific features.



(a) (b) (c)

Fig. 5. A toy example of block propagation in the Bitcoin network. (a) The P2P network structure, an undirected graph. (b) The gossiping process. A solid blue arrow represents one-hop block propagation (advertise→get block→transmit), while a dotted black arrow represents only advertise. Number denotes the gossiping hop (0 for the block producer). (c) Block propagation in one hop.

IV. The Nakamoto Consensus Protocol and Variations

The Nakamoto consensus protocol is the key innovation behind Bitcoin [1] and many other established cryptocurrency systems such as Ethereum [63] and Litecoin [64]. In this section we use Bitcoin as the application background to introduce the Nakamoto consensus protocol and summarize its drawbacks and vulnerabilities. We also introduce two well-known improvement proposals and four hybrid PoW-BFT protocols in the later part of this section.

A. Network Setting and Consensus Goal

In blockchain networks, block or transaction messages are propagated across the P2P network through gossiping. Figure 5 shows an example of block propagation in the Bitcoin network. Specifically, the one-hop propagation adopts advertisement-based gossiping, as was first characterized in [65]. For each new block received and validated, a node advertises it to its peers, who will request for this block if it extends it local blockchain. This gossiping process continues until every node in the network possesses this block.

Compared to the general consensus goal we introduced in Section III, Nakamoto consensus further specifies the termination requirement into probabilistic finality:

- **Probabilistic finality** For any honest node, every new block is either discarded or accepted into its local blockchain. An accepted block may still be discarded but with an exponentially diminishing probability as the blockchain continues to grow.

The probabilistic finality property echoes the probabilistic termination property for asynchronous consensus. As we will show later, because of this property the Nakamoto consensus protocol can only achieve eventual double-spending resistance in a decentralized network of pseudonymous participants.

B. The Nakamoto Consensus Protocol

In correspondence to the five components of a blockchain consensus protocol, the Nakamoto consensus protocol can be summarized by the following rules:

- *Proof of Work (PoW)*: Block generation requires solving a time-consuming hashing puzzle (i.e. mining). The puzzle difficulty is dynamically adjusted to maintain an average block generation time.
- *Gossiping rule*: Any newly received or locally generated transaction or block should be immediately broadcast to peers.
- *Validation rule*: A block or transaction need to be validated before being appended to the blockchain or being broadcast to peers. The validation includes double-spending check on transactions and proof-of-work validity check on block header.
- *Longest-chain rule*: The longest chain represents network consensus, which should be accepted by any node who sees it. Mining should always extend the longest chain.
- *Block rewards and transaction fees*: Generator of a block can claim a certain amount of new tokens (12.5 BTC at present) plus fees collected from all enclosed transactions, in the form of a *coinbase* transaction to herself.

The hashing-intensive PoW mechanism is designed for mitigating Sybil attacks. Due to Bitcoin's permissionless and pseudonymous nature, Sybil attackers can obtain new identities or accounts with little effort. Hashing power, however, comes from real hardware investment and cannot be easily forged. The longest-chain rule implies that the stabilized prefix of the longest chain can act as a common reference of the network history, given that no one is authoritative in Bitcoin's decentralized network. The block rewards and transaction fees are used to incentivize miners to participate honestly and inject new coins into circulation.

To better illustrate how these rules harmonize with each other, we present an abstracted version of the Nakamoto protocol in Algorithm 3. During block generation, a higher mining difficulty demands more brute-force trials in order to find a fulfilling nonce. To ensure every block is sufficiently propagated before the next block comes out, the protocol adjusts the mining difficulty every 2016 blocks so that the expected block interval remains a long-enough constant value (10 minutes in Bitcoin) no matter how the gross hashing power fluctuates.

Algorithm 3: Nakamoto consensus protocol general procedure

```
/* Joining network */
Join the network by connecting to known peers;
Start BlockGen();
/* Main loop */
while running do
    if BlockGen() returns block then
        Write block into blockchain;
        Reset BlockGen() to the current blockchain;
        /* Gossiping rule */
        Broadcast block to peers;
    end
    /* Longest-chain/validation rule */
    if block received & is valid & extends the longest
      chain then
        Write block into blockchain;
        Reset BlockGen() to the current blockchain;
        Relay block to peers;
    end
end
/* PoW-based block generation */
Function BlockGen():
    Pack up transactions (including coinbase);
    Prepare a block header context C containing the
      transaction Merkle tree root, hash of the last block
      in the longest chain, timestamp, and other essential
      information reflecting blockchain status;
    /* PoW hashing puzzle */
    Find a nonce that satisfies the following condition:

            Hash(C|nonce) < target

      wherein more preceding zero bits in target indicates
      a higher mining difficulty;
    return new block;
end
```

Fork resolution Ideally, the 10-minute block interval should be enough to ensure the thorough propagation of a new block so that no block of the same height is proposed. However due to the delay during the message propagation and the probabilistic nature of the hashing game, the possibility of two blocks of the same height be propagated in the network simultaneously can not be ignored. This situation is called a "fork" once detected by a node. Correspondingly, the longest-chain rule provides the criterion for fork resolution. Let's assume a miner receives two valid blocks B_1^k, B_2^k of the same block height k, then a fork is detected by this miner. It chooses B_1^k to continue (normally the first received) and may encounter the following cases:

- Case 1: If receiving or successfully generating a block B_1^{k+1} confirming B_1^k, accept B_1^k, B_1^{k+1} and "orphans" B_2^k.
- Case 2: If receiving a block B_2^{k+1} confirming B_2^k, switch to B_2^{k+1} and accept B_2^k, then orphans B_1^k.
- Case 3: If simultaneously receiving two blocks B_1^{k+1} and B_2^{k+2} confirming respectively B_1^k and B_2^k, choose one to follow and continue until case 1 or 2 is met.

Because of the randomized nature of PoW mining, the likelihood of encountering case 3 drops exponentially as time elapses, reflecting the probabilistic finality of Nakamoto consensus.

Security analysis In contrast to the classic distributed computing system whose fault tolerance capability is characterized by the number of faulty nodes the system can tolerate, fault tolerance of Nakamoto consensus is by characterized by percentage of adversarial hashing power the system can tolerate. It is proved by Garay et al. [66] that if the network synchronizes faster than the PoW-based block proposing rate, an honest majority among the equally-potent (in hashing power and bandwidth) miners can guarantee the consensus on an ever-growing prefix of the blockchain. The prefix represents the probabilistically stable part of the blockchain. In others words, as long as less than 50% of total hashing power is controlled by an adversary, the blocks produced by honest miners are timely propagated, the main chain contributed by the honest majority can eventually outgrow any malicious branch.

From the perspective of classical distributed consensus, the probabilistic finality of Nakamoto consensus cleverly circumvents the fundamental 1/3 BFT bound. In classical BFT consensus if more than 1/3 of population are malicious, the honest nodes will end up deciding conflicting values, leading to consensus failure. In Nakamoto consensus, however, conflicting decisions are allowed temporarily in the form of blockchain forks, as long as they will be eventually trimmed out by continuing effort of the honest majority. Therefore, the 1/3 BFT bound is not applicable to Nakamoto consensus and other blockchain consensus protocols designed for probabilistic finality.

As for double-spending resistance, assuming the adversary controls α fraction of the total hashing power and wishes to double-spend an output which is m blocks old, it needs to redo the PoW mining all the way from m blocks behind and grow a malicious chain fast enough to overtake the incumbent. The probability of this adversarial catch-up is $(\frac{\alpha}{1-\alpha})^m$ if $\alpha < 50\%$, which drops exponentially as m increases, again reflecting probabilistic finality. This probability equals to 1 if $\alpha \geq 50\%$. As a result, the 50% threshold is the safeguard behind Bitcoin's probabilistic finality, as well as resistance to double-spending and transaction history tampering.

C. Drawbacks and Vulnerabilities of Nakamoto Consensus

1) Tight tradeoff between performance and security: The Nakamoto consensus is widely criticized for its low transaction throughput. For instance, Bitcoin can process up to 7 transactions per second meanwhile the VISA payment network processes 2500 per second on average [67]. The reason behind Nakamoto consensus protocol's limited performance is the security implication of two protocol parameters: block interval and block size, along with probabilistic finality. As we discussed previously, the 10-minute block interval ensures every new block is sufficiently disseminated before the next one comes out. Reducing the block interval increases the transaction capacity, but will leave new blocks insufficiently disseminated and causes more forks to happen, undermining the security of the main chain. Note that although any fork can be resolved given enough time, the higher the fork rate, the larger the portion of honest mining power is wasted, which enables a double-spending attacker to overthrow the main chain with less than 50% mining power (estimated 49.1% by Decker et al. [65] in 2013). On the other hand, increasing the block size (currently 1MB) has the same effect, since larger block sizes will increase the block propagation delays and lead to insufficient dissemination. According to the measurement study by Croman et al. [3] in 2016, given the current 10-min block interval the maximum block size should not exceed 4MB, which yields a peak throughput of 27 transactions per second.

2) Energy inefficiency: As of January 2019, an average Bitcoin transaction consumes 431 KWh of electricity which can power 15 U.S. households for a day [68]. This enormous energy consumption is directly caused by the PoW-based block proposing scheme of Nakamoto consensus. As Bitcoin network's gross mining capacity grows, the Nakamoto consensus protocol has to raise the mining difficulty to maintain the average 10-min block interval, which in turn encourages miners to invest into more mining equipment with higher hashing rates. This vicious cycle shall continue as Bitcoin gains more popularity. In response, the blockchain community has come up with various block proposing schemes such as proof of stake (PoS), proof of authority (PoA), proof of elapsed time (PoET) as energy-saving alternatives to PoW.

3) Eclipse attack: As was discussed above, the security of Bitcoin network relies on the hashing power and communication capability of honest miners. If a powerful attacker manages to dominate the in/outward communication between a victim miner and the main network (i.e. "eclipsing"), then the victim will no longer be able to contribute to the extension of main chain [69]. More specifically, assume the percentage of the hashing power controlled by the eclipse attacker, the eclipsed victims, and the remaining honest miners are α, ε, and $(1-\alpha-\varepsilon)$, then the attacker's mining power shall be amplified to at least $\frac{\alpha}{1-\varepsilon}$. If the attacker decides to exploit the eclipsed victims for growing the malicious chain, its mining power can be further enhanced up to $\alpha+\varepsilon$ [70]. As a result, a double-spending attack becomes viable for the eclipse attacker if $\alpha+\varepsilon > 50\%$.

Eclipse attack can be seen as an exploit of the permissionless peer-to-peer network and the underlying Internet wherein connections are subject to physical bottlenecks and adversarial influences. A general approach to counter eclipse attacks is to secure the communication channels and increase the connectivity and geographical diversity of the peer-to-peer connections.

4) Selfish mining: The 50% fault tolerance of Nakamoto consensus is built upon the assumption that all miners (both honest and malicious) strictly follow the broadcast rule that new blocks are broadcast immediately upon successful generation. If a malicious mining group withholds newly mined blocks and strategically publicize them to disrupt the propagation of blocks mined by honest miners, they can partially nullify the work of honest miners and amplify their effective mining power. This strategy is known as *selfish mining*. It was shown by Eyal et al. [71] that a selfish mining group can generate a disproportionately higher revenue than that from honest mining if the group's mining power surpasses a certain threshold θ, assuming the group has a certain communication capability measured by $\gamma \in [0,1]$, which is the fraction of honest nodes that will follow the malicious chain in case of forks. As a result, the selfish mining group can attract new miners and eventually outgrow the honest miners. Notably, this threshold approaches zero if the selfish mining group are able to convince all honest miners to follow the malicious chain (i.e. $\gamma \to 1$). It was also shown that by adopting a randomized chain selection strategy at honest miners, which is equivalent to setting $\gamma = 0.5$, the threshold can be raised up to 25%. A later work by Sapirshtein et al. [72] shows that an optimized selfish mining strategy can further enhance the selfish mining pool's effective mining power fraction from α to the upper bound $\frac{\alpha}{1-\alpha}$ (achievable when $\gamma = 1$).

Interestingly, selfish mining attack and eclipse attack have only happened to smaller blockchains such as Monacoin [73], but never to Bitcoin or other mainstream blockchains. This is likely due to two reasons. First, miners in established public blockchain networks actually care about the system's longevity and reputation, which can positively affect the exchange rate of the cryptocurrency and thus their mining revenue. Second, established blockchains tend to be better connected (reflected by the existence of dedicated mining pools and relay networks), which allows for an effective detection of any selfish mining and eclipse attack behavior.

5) Centralization risk and mining pools: According to the incentive mechanism of Nakamoto consensus, the mining revenue of a miner is proportional to its computing power. Since bitcoins can be traded for fiat currencies at exchanges, higher-earning miners have the financial advantage to purchase more efficient mining hardware, which consumes less joules per hash operation. Furthermore, higher-earning miners are often backed by large organizations that can direct huge capital into the mining business. As a result, small individual miners are either forced out of the game, or alternatively join in mining pools for stabler income. All members in a mining pool are synchronized and work on a common chain; any block reward is shared by all pool members. In fact, joining in a mining pool has become the dominant way of participation in major PoW-based blockchains.

The measurement study by Gencer et al. [74] in 2018 shows that throughout a one-year observation period, over 50% of the gross mining power was exclusively shared by eight mining pools in Bitcoin and five mining pools in Ethereum. Moreover, the empirical study by Kondor et al. [75] in 2014 shows that the wealth distribution among Bitcoin addresses has been converging to a stable exponential distribution, and the wealth accumulation of node is positively related to its ability to attract new connections, which is another advantage of established large miners.

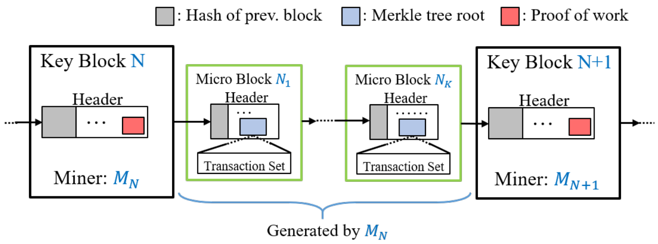


Fig. 6. Bitcoin-NG blockchain data structure.

D. Improvements to the Nakamoto Consensus Protocol

1) GHOST Rule: The greedy heaviest-observed subtree (GHOST) for Bitcoin rule was first proposed by Sompolinsky et al. [76] in 2015. A variation is implemented in the Ethereum blockchain platform. According to the longest-chain rule, all unconfirmed blocks in a fork shall be orphaned, resulting in a waste of honest mining power which could otherwise have been used to contribute to the longest-chain's security. The longest-chain rule also limits the transaction capacity because of the tight tradeoff between performance and security. The GHOST rule is an alternative to the longest-chain rule that the orphaned blocks also contribute to the main chain security, effectively reducing the impacts of forks, which allows for a shorter block interval and thus higher transaction capacity. Specifically, GHOST requires that given a tree of blocks with the genesis block as the root, the longest chain within the heaviest subtree shall be used as the main chain. Similar to the Nakamoto consensus, the probabilistic finality of the heaviest subtree up to the current block height will hold as long as more than 50% of mining power are honest. The simulation result in [76] shows that applying GHOST rule will lead to a slightly slower blockchain growth rate than that with the longest-chain rule, but it nearly perfectly prevents the security degradation when the block interval decreases, allowing for a higher transaction throughput.

Bitcoin-NG was proposed by Eyal et al. [7] in 2016 to scale up Bitcoin's transaction capacity. The key insight of Bitcoin-NG is decoupling block generation into two planes: *leader-election* and *transaction serialization*, which respectively correspond to two types of blocks: *key blocks* and *micro blocks*. The key blocks resemble Bitcoin's blocks, which contain a solution to a hash puzzle representing the proof of work and have an average block interval of 10 minutes, except for the actual transactions which are included in the micro blocks. Once a key block is mined, all subsequent micro blocks shall be generated by the current key block miner until the generation of the next key block. The generation of micro blocks is deterministic and does not contain proof of work. As a result, the micro block frequency is in the control of the key block miner (up to a maximum) to accommodate as many transactions as possible. The blockchain data structure of Bitcoin-NG is shown in Figure 6.

2) Bitcoin-NG: As for the consensus finality, the longest-chain rule is still applied to finalize and resolve forks of key blocks. As for the micro blocks, since they are batch-generated by key block miners, Bitcoin-NG relies on a combination of a heaviest-chain extension rule and a longest-chain extension rule to finalize and resolve forks of micro blocks. Notably, the micro block frequency needs to be controlled, as an excessive amount of them may exhaust the network bandwidth and cause more forks. Furthermore, to encourage honest participation and maximize the security against double-spending attacks, 60% of the transaction fees collected from micro blocks by the current key block miner are redistributed to the miner of the next key block. A variant of Bitcoin-NG called Waves-NG [77] is currently used in Wave Platform, a blockchain initiative.

On the negative side, because of the determinism in micro block generation, the key block miner may become a target of denial-of-service or corruption attacks. A compromised key block miner may selectively include transactions or finalize contradicting transactions, the inconsistency caused by which can take the network more than one key block cycle to remedy.

E. Hybrid PoW-BFT Consensus Protocols

The limited transaction capacity and tight tradeoff between performance and security of Nakamoto consensus are much warranted by its probabilistic finality and decentralized ideal. In contrast, BFT consensus assumes fixed participants with revealed identities and achieves deterministic finality, allowing much shorter block intervals and thus much higher transaction throughput. In response, several hybrid PoW-BFT protocols have been proposed to get the best of two worlds.

1) PeerConsensus: Proposed by Decker et al. [61] in 2014, PeerConsensus uses a PoW-based blockchain to throttle and certify new identities joining the network, while being agnostic to any application built upon it. The number of identities a player may control is proportional to its share of computation power, which provides Sybil resistance. With the identities established by the blockchain, the application can employ an efficient BFT protocol such as PBFT and SGMP [78] for committing transactions. The transaction fees collected are distributed to all identities equally. As a result, PeerConsensus effectively decouples participant management from transaction processing, allowing the latter to scale up throughput. On the flip side, since the transaction history is not recorded in blockchain, PeerConsensus cannot control the malleability of transactions.

2) SCP: The scalable consensus protocol (SCP), proposed by Luu et al. [79] in 2015, incorporates BFT and sharding into blockchain consensus. The key idea of SCP is to partition the network into sub-committees (i.e. shards) with a PoW mechanism, so that each sub-committee controls a limited amount of computation power and the number of sub-committees is proportional to the network's gross computation power. This is aimed to limit the size of a sub-committee, which operates independently and curates a local blockchain using a BFT consensus protocol. A designated finalization committee is responsible for combining the outputs of all sub-committees into the global blockchain. Specifically, a block in the global chain contains the hash and transaction Merkle tree root of the block proposed by each sub-committee. To ensure consensus safety, SCP requires each sub-committee and the whole network to maintain a two-thirds majority of honest computation power. However, the use of sharding and a dedicated inter-shard finalization committee implies the preexistence of network coordination, which to some extent counters the decentralized ideal of public blockchains.

Notably, using sharding to scale up a blockchain network's transaction capacity has been extensively studied in the developer's community. Interested readers are referred to [80] for a detailed development history and state-of-the-art of sharding techniques.

3) ByzCoin: Proposed by Kogias et al. [81] in 2016, ByzCoin is another blockchain consensus protocol that leverages PoW for consensus group membership management and BFT for transaction finalization. ByzCoin's ledger consists of two parallel blockchains: a keyblock chain and a microblock chain, an idea adapted from Bitcoin-NG's blockchain. A keyblock is mined via PoW as in Bitcoin and determines a consensus group according to a sliding-window mechanism on recent keyblock miners. A microblock is produced by the current consensus group via an adapted PBFT protocol based on collective signing (CoSi) [82]. Compared to the original PBFT, the CoSi-based PBFT achieves lower communication complexity and scales better to large consensus groups. As for transaction finalization, the current keyblock miner packs up new transactions into a microblock and acts as the leader to trigger the CoSi-based PBFT. In the end, the microblock will be finalized and contain the collective signature of the consensus group and the hash pointer to the preceding keyblock, which also contains the collective signature.

ByzCoin configures that the sliding-window mechanism can replace one member of the consensus group at a new keyblock. This yields a slightly tighter fault-tolerance than that of classical BFT consensus: $N \geq 3f+2$ is needed at any time, where N is the consensus group size and f is the Byzantine population. On the other hand, ByzCoin's PoW-based keyblock chain is still susceptible to forks. A fork can split the consensus group and potentially make the PBFT consensus stall, which can further aggravated by the presence of selfish miners. In response, ByzCoin relies on a deterministic prioritization function tweaked with high output entropy to resolve forks timely and reduce the impact of selfish miners.

4) Pass and Shi's hybrid consensus: As a concurrent work to ByzCoin, the hybrid consensus protocol proposed by Pass and Shi [83] adopts a sliding-window idea similar to ByzCoin's but less susceptible to forks. That is, assuming the window size λ, the consensus group is populated by the last λ miners of the "stable part" of the blockchain, which is the main chain truncated off $\Theta(\lambda)$ blocks. This protocol keeps the PBFT consensus off-chain; only the consensus epoch number and a digest of the transaction log are attached to the new block. Moreover, this protocol advocates using FruitChain [84] as the underlying PoW blockchain, which was proposed by the same authors and can allegedly achieve better ledger tamper-resistance than Nakamoto's blockchain.

V. Proof-of-Stake Based Consensus Protocols

Proof-of-Stake (PoS) originates from the Bitcoin community as an energy efficient alternative to PoW mining. In

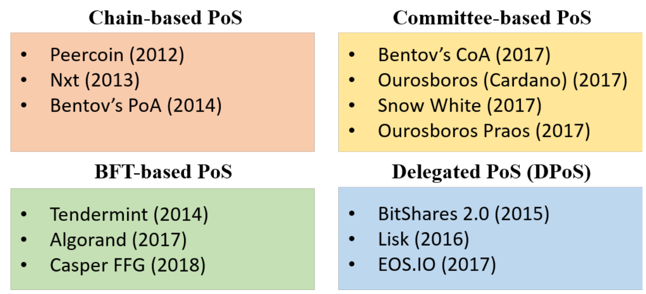


Fig. 7. Popular PoS blockchain initiatives, classified under four classes.

the simplest terms, a stake refers to the coins or network tokens owned by a participant which can be invested in the blockchain consensus process. From the security point of view, PoS leverages token ownership for Sybil attack mitigation. Compared to a PoW miner whose chance to propose a block is proportional to its brute-force hashing power, the chance to propose a block of a PoS miner is proportional to the value of its stake. From the economics point of view, PoS moves a miner's opportunity cost from outside the system (waste of hashing power & electricity) to inside the system (loss of capital & investment gain) [85]. Because of the lack of real mining, we often refer to a PoS miner as a *validator*, *minter*, or a *stakeholder* for PoS's close resemblance to investing in capital markets.

We identify four classes of PoS protocols: *chain-based PoS*, *committee-based PoS*, *BFT-based PoS*, and *delegated PoS* (DPoS). Chain-based PoS inherits many of the features of the Nakamoto consensus protocol such as information propagation, block validation, and block finalization method (i.e. longest-chain rule), except that the block generation method is replaced with PoS. Committee-based PoS leverages a multiparty computation (MPC) scheme to determine a committee to orderly generate blocks. BFT-based PoS combines staking with tradition BFT consensus which guarantees deterministic finality of blocks. DPoS enploys a social voting mechanism that elects a fixed size group of delegates for transaction validation and blockchain consensus on behalf of small stakeholders. Popular blockchain initiatives for each PoS class are listed in Figure 7.

A. Chain-based PoS

Chain-based PoS is an early PoS scheme proposed by Bitcoin developers as an alternative block generation mechanism to PoW. It is within the framework of Nakamoto consensus in that gossiping-style message passing, the block validation rule, the longest-chain rule, and the probabilistic finality are preserved. Early full-fledged chain-based PoS blockchain systems include Peercoin and Nxt.

The general procedure of a chain-based PoS minter can be summarized by Algorithm 4. Unlike PoW, PoS does not hinge on wasteful hashing to generate blocks. A minter can solve the hashing puzzle only once for a clock tick. Since the hashing puzzle difficulty decreases with the minter's stake value, the expected number of hashing attempts for a minter to solve the puzzle can be significantly reduced if her stake value is high.

Algorithm 4: Chain-based PoS general procedure (Peercoin, Nxt)

```
Join the network by connecting to known peers;
Deposit in the stake pool;
Start BlockGen();
/* Main loop */
while running do
    (Omitted here. Same with Nakamoto's protocol
     except that block validation should include PoS
     check.)
end
/* PoS-based block generation */
Function BlockGen():
    Pack up transactions and prepare a block header
     context 𝒞 containing the transaction Merkle tree
     root and other essential blockchain information;
    /* PoS hashing puzzle */
    Set up a clock (whose tick interval is a constant) and
     check for the following condition per clock tick:

         Hash(𝒞|clock_time) < target × stake_value

     wherein more preceding zero bits in target indicates
     a higher mining difficulty per unit of stake value;
    return new block;
end
```

Therefore, PoS avoids the brute-force hashing competition that would have taken place had PoW been used, thus achieves a significant reduction of energy used for generating a block.

1) Peercoin and Nxt: Both Peercoin [6] and Nxt [86] generally follow Algorithm 4. Their major difference lies in how the stake is valued. Generally stake value is proportional to stake quantity. To ensure the profitability of small stakeholders, a stake valuation scheme can be used to adjust the value of an unused stake as time passes. Peercoin uses the *coin age* metric for stake valuation, which means the value of a stake appreciates linearly with time since the deposit. At the end of a block cycle, the value of the winner's stake returns to its base value. To avoid stakeholders from locking in a future block by deliberately waiting long, stake appreciation only continues for 90 days and stays flat since then. As a result, the chances of small stakeholders to generate a block are supplemented with time value that encourages them to stay participated even if they haven't generated a block for a long time.

In comparison, Nxt does not appreciate stake value continuously across block cycles like what Peercoin does. This is because the latter's coin age metric may lower the attack cost—attackers can just invest a small amount of stakes and keep attempting to generate blocks until they succeed, which is especially unfair to big stakeholders who perform honestly. Instead, Nxt requires the stake value appreciate only within one block cycle and be reset to the base value once the block cycle ends. Without coin age, Nxt addresses the monopolization problem from the incentive angle. First, stakes in Nxt are not actively managed by stakeholders; they are essentially the account balances—the more tokens held in its account, the higher the chance the stakeholder will win the right to generate a block. Second, total token supply is determined at the beginning and block rewards only come from transaction fees, which effectively aligns a stakeholder's revenue with its validation work. As a result, all stakeholders have the incentive to honestly validate transactions since it is the only way to accumulate wealth.

2) Bentov's PoA: Compraring to Peercoin and Nxt, Bentov's proof of activity (PoA) [87] employs a rather innovative method to put stakes in the loop. Bentov's PoA is a hybrid PoW-PoS adaptation of the Bitcoin protocol that utilizes an algorithm called *follow-the-satoshi* (FTS) to involve stakes. FTS works as follows: 1) Use a pseudo-random function (PRF) to locate an atomic piece of token (eg. satoshi in Bitcoin, wei in Ethereum) in the token universe; 2) If the atomic piece belongs to stakeholder A, then output A. With the input of a list of random seeds, FTS can output a pseudo-random sequence of stakeholders such that chance of a stakeholder being in the sequence is proportional to the volume of tokens owned by the stakeholder.

Bentov's PoA works as follows. At the beginning of block cycle k, an empty block header EB_k is generated according to the PoW rule and propagated across the network. After receiving EB_k, a stakeholder computes the N-dimension vector seed S as follows:

$$S_j = hash\Big(hash(EB_k)|hash(B_{k-1})|SF_j\Big) \quad \text{for } j = 1, \ldots, N$$

B_{k-1} is the previous block, SF is a N-tuple of fixed suffix values. N is a predefined value that should not be too large. Then S is used as the input for FTS to compute the pseudo-random sequence of stakeholders *pSeq*. Every stakeholder in *pSeq* needs to sign the block and broadcast the signature; the last stakeholder in *pSeq* wraps up the block by including transactions and the N signatures (including hers) and broadcasts the final block B_k to the network. All stakeholders in *pSeq* will share the reward of B_k with the PoW miner of EB_k.

To avoid name conflict with proof of authority, we may also refere to Bentov's PoA as PoAct.

Security analysis Chain-based PoS can tolerate up to 50% of all stakes being maliciously controlled. And since every token can be staked, the fault tolerance further generalizes to 50% of all tokens in the network. If a colluding group control more than 50% of stakes, they can grow their malicious chain faster than the others and carry out a double-spending attack, which is analogous to the 51% attack in PoW blockchains. However, from the economic perspective, PoS attackers have lower incentives to do so because of the capital loss risk. As staking is recorded in the form of transaction scripts, the blockchain users can retrieve the staking records from which the consensus protocol can legally issue punishment to violators, such as nullifying stakes and disbarring the violators from participating in the future staking process.

B. Committee-based PoS

Chain-based PoS still relies on the hashing game to generate blocks. As an alternative mechanism, committee-based PoS allows for a more efficient regime: determining a committee of

Algorithm 5: Committee-based PoS general procedure

```
/* Joining network and staking */
Join the network by connecting to known peers;
Deposit in the stake pool;
/* Main loop */
while running do
    /* Committee election */
    if new block cycle then
        Participate in CommitteeElect();
        Check BlockGenSeq for my turns;
    end
    /* Block proposing & broadcast */
    if my turn to generate block then
        Collect transactions and generate block;
        Write block to blockchain;
        Broadcast block to the network;
    end
    /* Longest-chain&validation rule */
    if block is received & is valid & extends the longest
      chain then
        Write block into blockchain;
        Relay blocks to other committee members;
    end
end
/* PoS-based committee election */
Function CommitteeElect():
    Fetch the current blockchain state and the stake
      information of all participants; use them as the
      MPC input;
    Participate in the MPC that produces BlockGenSeq, a
      pseudo-random sequence of block generation
      opportunities;
    return BlockGenSeq;
end
```

stakeholders based on their stakes and allowing the committee to generate blocks orderly. A secure *multiparty computation* (MPC) scheme is often used to derive such a committee in the distributed network. MPC is a genre of distributed computing in which multiple parties beginning with individual inputs shall output the same result [88]. The MPC procedure in the committee-based PoS essentially realizes a verifiable random function (VRF) that takes in the current blockchain state and the stake values from all stakeholders, and outputs a pseudo-random sequence of stakeholders (we call it the *leader sequence*) which will sequentially populate the block-proposing committee. This leader sequence should be the same for all stakeholders and those with higher stake values may take up more spots in the sequence. A general procedure for a stakeholder of committee-based PoS is shown in Algorithm 5. Note that *CommitteeElect()* can also be implemented in a privacy-preserving way that only the stakeholder itself knows if it is elected.

Well-known committee-based PoS schemes include Bentov's chain of activity (CoA) , Ouroboros (for Cardano), Snow White, and Ouroboros Praos. Interestingly, these protocols and

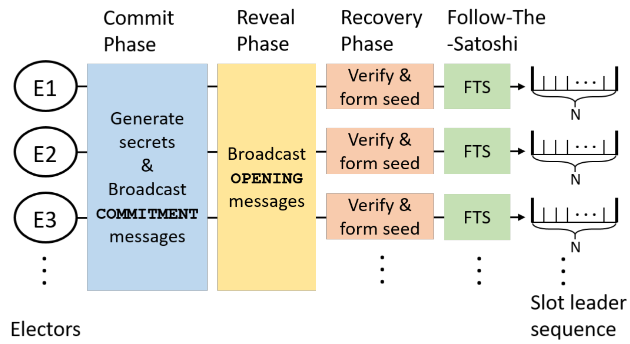


Fig. 8. MPC for the next leader sequence in Ouroboros. Secrets generated by electors at the beginning of the commit phase eventually form the random seed string. All electors shall go through FTS and obtain the same slot leader sequence at the end.

Algorand (see Section V-C) were developed concurrently by academics around the year 2017 and share many common traits.

1) Bentov's CoA: Bentov's CoA [89] was proposed in 2016 based partly on Bentov's PoA. It follows the main routine in Algorithm 5 in that each nominated stakeholder gets to generate its own block. CoA first leverages a MPC process to generate a string S of N random bytes. Then S is fed to a FTS algorithm that outputs a pseudo-random sequence *BlockGenSeq*. All parties should output the same *BlockGenSeq*, which is then used to coordinate the generation of the next N blocks.

2) Ouroboros: Ouroboros was developed by Kiayias et al. [8] in 2017 and used as the consensus protocol for cryptocurrency Cardano. It is similar to Bentov's CoA but with finer time arrangement. Ouroboros divides the physical time into fixed-time epochs and each epoch is divided into N slots, each can be used by only one slot leader to generate a block for the network. For each epoch, stakeholders with enough stake ($\geq 2\%$ circulating tokens) can become electors, who will collectively elect slot leaders (i.e. the committee) for the next epoch through a MPC procedure, as is shown in Figure 8. In the commit phase, elector E_i broadcasts a commitment message that includes a random secret. In the reveal phase, E_i broadcasts an opening message that reveals the previously sent secret. In the recovery phase every elector verifies that commitments and openings match and then form a seed string with the revealed secrets. All electors have the same seed string and shall obtain the same slot leader sequence after executing FTS.

3) Ouroboro Praos: Ouroboros Praos was proposed by David et al. [90] in 2017 to address two security concerns of Ouroboros. First, Ouroboros requires stringent network synchrony for slot leaders to use their allocated slots precisely, which is vulnerable to desynchronization attacks. In comparison, Ouroboros Praos is designed for partially synchronous networks wherein a maximum delay of Δ slots is allowed for message delivery, albeit Δ is unknown to electors. This is achieved by strategically allocating empty slots that help electors to re-synchronize to the network. Second, Ouroboros is susceptible to adaptive corruption against slot leaders. Since the leader sequence for the next epoch is known to all network participants ahead of the actual block proposing, an attacker may plan to corrupt or compromise targeted slot leaders. In comparison, Ouroboros Praos adopts a special designed locally executed VRF scheme that allows each elector to know her block proposing slots for the next epoch secretly and generate the corresponding VRF proofs. Compared to Ouroboros' MPC, this scheme saves much of communication overhead at the cost of local cryptographic computation. Similar schemes are also used in contemporary protocols such as Snow White and Algorand. In addition to the VRF scheme, Ouroboros Praos also adopts key-evolving signatures (KES) to counter posterior corruption and provide forward security, which we will elaborate in Section V-E.

4) Snow White: Snow White was developed by Daian et al. [9] in 2017. It is a PoS protocol specifically designed to accommodate the sporadic participation model in which nodes can switch online/offline arbitrarily. Similar to the committee-based PoS schemes above, Snow White employs a MPC procedure to decide the block proposing committee, each of which is issued an eligibility ticket privately. For each epoch, every stakeholder takes the current blockchain (including staking information) as input and output the committee for the next epoch. Specially, Snow White executes a modified version of the sleepy consensus protocol [91], an asynchronous consensus protocol that ensures the consensus safety in case of sporadic participation and committee reconfiguration. Comparing to its contemporaries (including Algorand), this feature enables Snow White to work under harsh network conditions such as frequent disconnection and unstable message delay. Moreover, Snow White uses a checkpointing scheme to finalize earlier history that protects the blockchain from posterior corruption attack [87] and adaptive key selection attack.

Security analysis In spite of having a more orderly block proposing scheme, committee-based PoS still sticks with the longest chain rule for probabilistic finality. Therefore as long as less than 50% stakes are held by malicious parties, the honest majority can safely maintain a correct chain. However, as we have discussed about Ouroboros, the viability (or liveness) of the MPC process largely relies on the network size and connectivity. The deterioration in network connectivity may result in a significant drop in protocol performance and fault tolerance. The committee election process driven by synchronous rounds (i.e. Snow White) also faces scalability problems, as large committee sizes may lead to never ending consensus cycles.

C. BFT-based PoS

Chain-based PoS and committee-based PoS largely follow the Nakamoto consensus framework in that the longest-chain rule is still used to provide probabilistic finality of blocks. In comparison, BFT-based PoS (or hybrid PoS-BFT) incorporates an extra layer of BFT consensus that provides fast and deterministic block finalization. Algorithm 6 shows the general procedure of BFT-based PoS at every participant. Block proposing can be done by any PoS mechanism (PoW, round-robin, committee-based, etc.) as long as it injects a stable flow of new blocks into the BFT consensus layer.

Aside from the general procedure, a checkpointing mechanism can be used to seal the finality of the blockchain (not shown in Algorithm 6). As a result, the longest-chain rule can be safely replaced by the most-recent-stable-checkpoint rule for determining the stable main chain. Popular BFT-based PoS blockchain protocols include Tendermint, Algorand, and Casper FFG. DPoS protocols such as EOS.IO also use BFT consensus for block finalization within delegates.

Algorithm 6: BFT-based PoS general procedure

```
Join the network by connecting to known peers;
Start BlockGen();
/* Main loop                                  */
while running do
    /* Block proposing & broadcast            */
    if BlockGen() returns block then
        Add block to its tempBlockSet;
        Broadcast block to the network;
    end
    /* Block validation                       */
    if block is received & is valid then
        Add block to its tempBlockSet;
        Relay block to the network;
    end
    /* BFT consensus layer                    */
    if new consensus epoch then
        Perform BlockFinBFT() on tempBlockSet;
        Write the winning block to blockchain;
        Clear tempBlockSet;
    end
end
/* PoS-based block generation                 */
Function BlockGen():
    (Any feasible PoS mechanism that injects a stable
     flow of blocks to the BFT consensus layer.)
    return block;
end
/* BFT-based block finalization               */
Function BlockFinBFT():
    Participate in a BFT consensus that finalizes one
     winning block out of tempBlockSet;
    return the winning block;
end
```

1) Tendermint: Tendermint was develop by Kwon [12] in 2014 inspired by the DLS protocol [27] and PBFT [26]. Tendermint works in consensus cycles, each involves a multi-round BFT consensus process to finalize one block. Each round consists of three steps: propose, prevote, precommit. Specially, in the propose step a validator is designated by a deterministic algorithm as the block proposer in a round-robin fashion such that validators are chosen with frequency proportional to the value of their deposited stakes. The rounds carry on until one block receives more than 2/3 of commit votes, which then becomes the block to be finalized in blockchain. Notably, since Tendermint decouples stake value from BFT votes, as long as more than 2/3 of validators are honest, Tendermint can always achieve correct consensus. On the other hand, because Tendermint selects the round-robin block proposers deterministically, the future block proposers are susceptible to DDoS attacks. Tendermint addresses this DDoS risk by deploying sentry nodes which act as proxies of block proposers and never reveal the IP addresses of the latter.

2) Algorand: Algorand is a cryptocurrency system developed by Gilad et al. [13] at MIT CSAIL in 2017. It employs committee-based PoS for block proposing and Byzantine agreement for block finalization. First, the election of the block proposing committee is done by a scheme called *cryptographic sortition*, a randomized algorithm that sorts candidates according to the amount of coins they own, via a mechanism similar to FTS. Only those with rankings above a threshold are admitted into the committee for the next block cycle. Every individual user can check privately if it is in the committee. At each user i, cryptographic sortition also outputs an eligibility proof signed by the user's private key $\sigma_{sk_i}^{ep}$ showing that it is truly a committee member. $\sigma_{sk_i}^{ep}$ is broadcast to the network along with the new block proposed by user i. Upon receiving the block, other users can verify the proof via the user i's public key pk_i.

On top of the cryptographic sortition-based block proposing scheme, Algorand relies on a Byzantine agreement protocol called $BA\star$ for block finalization. $BA\star$ reduces the consensus problem to binary Byzantine agreement: either agreeing on a proposed block or an empty block. In the ideal case where strong network synchrony is assumed, the committee follows $BA\star$ to exchange votes on proposed blocks so that they will decide a final block, or an empty block if no blocks pass the eligibility proof check. In a weakly synchronous network where block propagation and message exchange among committee members can suffer from uncertain delays, $BA\star$ outputs tentative blocks if none of the proposed blocks can be finalized, which results in a fork. To resolve the forks of tentative blocks, Algorand periodically runs a recovery protocol to accept a tentative if there is any. Specially, the recovery protocol needs to be invoked in a synchronous fashion across the committee members. Therefore, as is mentioned in the Alogrand paper [13], weak synchronony is sufficient for consensus safety during $BA\star$'s routine operation while strong synchrony is required for consensus liveness when kicking off the recovery protocol for fork resolving.

3) Casper FFG: Casper FFG [14] was developed in 2017 and purposed as the first step of Ethereum's transition to PoS. It is a light-weight consensus layer that can be used on top of Ethereum's current PoW-based block proposing mechanism (Ethash). Casper FFG slightly deviates from Algorithm 6 for that it directly builds PoS into block finalization.

Algorithm 7 shows the general procedure of Casper FFG at a validator. Newly generated and received blocks are attached to the *BlockTree*, which is similar to the block tree used by the GHOST rule. The actual consensus subject, however, is *CheckPointTree*, which is a subtree of *BlockTree*. Specifically, for every consensus epoch (100 in *BlockTree*'s height or 1 in *CheckPointTree*'s height), every validator broadcasts to peers a vote for a block as the checkpoint. The height of the block in *BlockTree* must be divisible by 100. The vote consists

Algorithm 7: Casper FFG

Deposit in the stake pool;
`/* Main loop */`
while *running* **do**
  (Block proposing and block validation are the same as in Algorithm 6, except that blocks are attached to *BlockTree* rather than stored in a temporary set.)
  `/* BFT consensus layer */`
  if *new consensus epoch* **then**
    Identify valid checkpoint blocks and attach them to *CheckPointTree*;
    Participate in *CheckPointVote()* w.r.t. *CheckPointTree*, which returns CP_s, CP_t;
    Mark CP_s *finalized* and CP_t *justified*;
  end
end
`/* Staked checkpoint voting */`
Function *CheckpointVote()***:**
  Broadcast a vote for a source-target checkpoint pair in *CheckPointTree*;
  Check received votes against the slashing rules and then evaluate them by signer's deposited stake;
  if *pair* $\langle CP_s, CP_t \rangle$*'s votes cover more than 2/3 of total deposited stakes* **then**
    return CP_s, CP_t;
  end
end

of a *justified* source checkpoint CP_s and its height $h(s)$, a target checkpoint CP_t and its height $h(t)$ ($h(s) < h(t)$), and the validator's signature S. All votes are broadcast to the network and are weighted by the signer's stake value. If the source-target checkpoint pair $\langle CP_s, CP_t \rangle$ are voted by validators who possess more than 2/3 of total deposited stakes, then CP_t is justified and CP_s is *finalized*. All blocks between CP_s and CP_t are finalized as well. Casper FFG relies on two so called *Casper Commandments* for ensuring consensus safety: 1) a validator must not cast two distinct votes for the same checkpoint height, and 2) a validator must not cast a new vote whose source-target span is within that of its existing vote. Violators are subject to slashing rules including forfeiting stakes and temporarily banning from staking. Since every vote is signed with the validator's private key and received by peer validators, Casper FFG can conveniently detect violators and enforce the slashing rules.

The current smart contract implementation of Casper FFG is documented in EIP 1011 [62]. A stakeholder becomes a Casper participant (also called a validator) by depositing a stake in the dedicated smart contract and interacts with it via transactions.

Security analysis BFT-based PoS guarantees consensus safety if fewer than 1/3 of total stakes are maliciously controlled. In Algorand, if an attacker group own more than 1/3 of total tokens, then the probability is high that more than 1/3 of the elected committee members belong to the attacker group, which results in the consensus failure of $BA\star$.

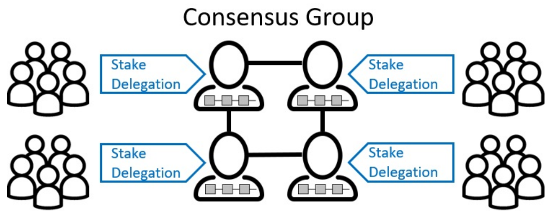


Fig. 9. Example of the delegation process in DPoS. Small stakeholders vote for delegates with their stakes and the most voted delegates form the consensus group.

In Casper FFG, if an attacker group own more than 1/3 of total deposited stakes and dominate the communication within the network, they can vote on conflicting checkpoints without getting punished. Additionally, since a typical BFT consensus protocol can utilize a checkpointing mechanism to ensure deterministic finality of the block, satisfying the 1/3 fault tolerance bound automatically fends off costless simulation attacks (to introduce in Section V-E). Note that breaking the 1/3 threshold only breaks the guaranteed safety of the BFT consensus; to dominate the blockchain finalization the attackers need to control more than 2/3 of all stakes.

D. Delegated PoS (DPoS)

DPoS can be seen as another form of committee-based PoS, only the committee (consensus group) is chosen via public stake delegation. It is currently used by BitShares 2.0 (2015) [92], Lisk (2016) [93], and EOS.IO (2017) [10]. DPoS was designed to control the size of the consensus group so that the messaging overhead of the consensus protocol remains manageable. Members of the consensus group are also called *delegates*. The election of delegates is called the delegation process, a general example of which is shown in Figure 9. In the actual case, the delegation process (and the soliciting of votes) may involve outside incentives.

In EOS.IO for example, the delegation process turned out to be an interesting socioeconomic phenomenon. An aspirating delegate needs to attract enough votes from small token holders. This is often accomplished by offering a killer application or building up reputation through propaganda campaigns. By casting a vote via a blockchain transaction, a token holder entrusts the delegate with its own stake. As a result, the delegate harvests the stake voting power from its voters and acts as their proxy in the consensus process. The EOS.IO protocol mandates that the top 21 voted delegates are admitted into the consensus group wherein the right of block generation is equally shared in a round robin fashion. Specially, EOS.IO employs a pipelined PBFT-style consensus scheme to finalize the proposed blocks across the 21 delegates [94]. As a result of this orderly procedure, every new block can be immediately finalized by all delegates and recognized by their voters.

Security analysis Assuming BFT is used by the consensus group for block finalization, which is recommended since the group size is limited, DPoS can tolerate 1/3 of delegates being malicious. For example, EOS.IO can tolerate at most 6 out of 21 delegates being malicious. In the real world they may not want to misbehave or collude at all, since all delegates have revealed identities to voters and would be scrutinized for any misconduct.

E. Vulnerabilities of PoS

Although heralded as the most promising mechanism to replace PoW, PoS still has several vulnerabilities that need to be addressed before wide adoption.

1) Costless simulation: Costless simulation is a major vulnerability of non-BFT-based PoS schemes, especially chain-based PoS wherein PoS is used to simulate the would-be PoW process. Costless simulation literally means any player can simulate any segment of blockchain history at the cost of no real work but speculation, because PoS does not incur intensive computation while recording all staking history on the blockchain. This may give shortcuts to attackers to fabricate a fake blockchain. The four subsequent vulnerabilities, namely *nothing-at-stake*, *posterior corruption attack*, *long-range attack*, and *stake-grinding attack* are all based on costless simulation.

2) Nothing-at-stake: Nothing-at-stake is the first identified costless simulation problem that affects chain-based PoS. It is also known as “multi-bet” or “rational forking” problem. Unlike a PoW miner, a PoS minter needs little extra effort to validate transactions and generate blocks on multiple competing chains simultaneously. This “multi-bet” strategy makes economical sense to PoS nodes because by doing so they can avoid the opportunity cost of sticking to any single chain. Consequently if a significantly portion of nodes perform the “multi-bet” strategy, an attacker holding far less than 50% of tokens can mount a successful double spending attack [95]. Nothing-at-stake problem can be practically solved by penalizing whoever multi-bets, such as forfeiting part of or all their stakes. However the penalties could still be reversed if the attacker eventually succeeds in growing a malicious chain.

3) Posterior corruption: Dubbed by Bentov [87] as “bribing attack” in 2014, posterior corruption is another attack utilizing costless simulation against PoS. The key enabler of posterior corruption is the public availability of staking history on the blockchain, which records stakeholder addresses and staking amounts. An attacker can attempt to corrupt the stakeholders who once possessed substantial stakes but little at present by promising them rewards after successfully tampering with the transaction history. When there are enough stakeholders corrupted, the colluding group (attacker and corrupted once-rich stakeholders) could own a significant portion of tokens (possibly more than 50%) at some point in history, from which they are able to grow an alternative chain that will eventually surpass the current main chain. Since posterior corruption is only possible because the private/public keys are fixed once for all, key-evolving cryptography (KEC) [96] can be applied so that the past signatures cannot be forged by the future private keys. Ouroboros Praos [90] currently adopts KEC for this purpose. Alternatively, as is used in Snow White [9] and Casper FFG [14], checkpointing can be used to finalize the ledger and eliminate the possibility of posterior corruption.

4) Long-range attack: Coined by the Ethereum founder Vitalik Buterin [97], long-range attack can be viewed as the ultimate form of costless simulation. It foresees that a minor group of colluding attackers can regrow a longer valid chain that starts not long after the genesis block. Because there were likely only a few stakeholders and a lack of competition at the nascent stage of the blockchain, the attackers can grow the malicious chain very fast and redo all the PoS blocks (i.e. by costless simulation) meanwhile claiming all the historical block rewards. If the targeting blockchain network is not incentivized by block rewards, the attacker can still deploy a similar long-range scheme called *stake-bleeding attack* [98] as long as transaction fees exist. That is, the attacker group can accumulate significant amount of wealth by growing the malicious chain that collects nearly all historical transaction fees. Through either scheme, once the malicious chain overtakes the honest chain, it is released to public and becomes the longest chain.

While zero transaction fees can counter stake-bleeding attacks, general long-range attacks (and costless simulation as a whole) can be resolved by a more radical measure—checkpointing. Checkpointing is widely used in BFT protocols to ensure the finality of system agreements and safely discard older records. For permissionless blockchains (the main venue for chain-based PoS), however, checkpointing can undermine decentralization, as the finality of checkpoints always requires the endorsement from certain authoritative entities.

5) Stake-grinding attack: Generally, the block generation competition in Nakamoto-style blockchain with proof-of-X block proposal is a pseudo-random process that a higher X (work, stake, etc.) yields a higher probability of winning the competition. However, unlike PoW in which pseudo-randomness is guaranteed by the brute-force use of a cryptographic hash function, PoS's pseudo-randomness is influenced by extra blockchain information—the distribution of stakes. As a consequence, malicious PoS minters may take advantage of costless simulation and other staking-related mechanisms to bias the randomness of PoS in their own favor, thus achieving higher winning probabilities compared to their stake amounts [8]. For example, in chain-based PoS blockchains such Peercoin, attackers can iterate through different block headers that would increase the probability of their stakes generating a valid block [99]. For committee-based and BFT-based PoS schemes that decouple the election of block proposers from block generation, stake-grinding attacks can be mitigated by ensuring the PoS pseudo-randomness through a secure block proposer election scheme that involves minimal local information. Examples include Ouroboros, Ouroboros Praos, and Algorand.

6) Centralization risk: PoS faces a similar wealth centralization risk as with PoW. In PoS the minters can lawfully reinvest their profits into staking perpetually, which allows the wealthy become wealthier and eventually reach a monopoly status. When a player owns more than 50% of tokens in circulation, the consensus process will be dominated by this player and the system integrity will not be guaranteed. Take Ethereum's Casper FFG for example, the proposed PoS scheme is built upon the current PoW system, of which the cryptocurrency ethers can be directly used for staking. This gives initial advantages to those who have already accumulated huge wealth during Ethereum's PoW era. Potential countermeasures against monopolization in PoS mainly come from the economic perspective. In addition to the stake valuation scheme that improves the winning chances of small stakeholders (see Section V-A), we can introduce off-chain factors to complicate the staking process (EOS.IO for example) and instate taxation on the blocks generated by large stakeholders, to name a few.

VI. Other Emerging Blockchain Consensus Protocols

The majority of contemporary blockchain systems build their consensus protocols from the options of PoW, PoS, and BFT. There are also many other promising consensus protocols for specific use cases, many of which are still conceptual and may cover just one or two of the five blockchain consensus protocol components. In this section we present six of such protocols: proof of authority (PoA), proof of elapsed time (PoET), proof of TEE-stake (PoTS), proof of retrievability (PoR), Ripple consensus protocol/algorithm (RCPA), and IOTA Tangle.

A. Proof of Authority (PoA)

Proof of authority was coined by Ethereum co-founder Gavin Wood as an alternative to PoW and PoS. It is currently deployed in Ethereum's Rinkeby (2017) and Kovan (2017) testnet, and POA Network (2018) [11]. To avoid name conflicts, in later comparisons we refer to Bentov's PoA as PoAct, and proof of authority as PoA. In a nutshell, PoA is a special case of PoS in that a validator stakes with its identity instead of monetary tokens. To qualify as a PoA validator and join the consensus group, a participant needs go through a nontrivial process to build up its authority. It generally involves having the unique identity verified, demonstrating the ability to contribute consistently to the consensus, and making all certification documents publicly available. Therefore, the consensus group should be stable, small in size, and publicly scrutinized so that users can entrust the consensus group for reliably processing transactions and maintaining the blockchain. If a validator misbehaves or shows signs of incompetence, it will be discredited by users and peer validators.

Security analysis The fault tolerance of a PoA blockchain depends on the consensus protocol used by the consensus group. Besides the 1/3 fault tolerance of a BFT protocol, Nakamoto-style protocols such as Parity's AuthorityRound (AuRa) [100] can tolerate up to 50% of colluding validators. Featured by its small but trusted consensus group, PoA is a good example of compromising on decentralization for security and performance. To prevent validators from collusion, they should operate independently from each other and be constantly monitored by users. PoA is currently used in Ethereum's Rinkeby/Kovan testnet and POA Network [11].

B. Proof of Elapsed Time (PoET)

PoET was proposed by Intel as an alternative mining mechanism in 2016 and subsequently used in the Hyperledger Sawtooth family [101]. Instead of undergoing the hashing-intensive mining, PoET simulates the time that would be consumed by PoW mining. In other words, every node randomly backs off for an exponentially distributed period of time before announcing its block. To ensure that the local time truly elapses, PoET requires the back-off mechanism to be carried out in a trusted execution environment (TEE), which is an isolated memory area that provides integrity and confidentiality to the program running inside, even against the hosting platform. Intel SGX [102] and Arm Trustzone [103] are the two major TEE solution providers. Specially, the program enclosed in a TEE is called an "enclave". Among other utilities, TEE can provide an integrity proof of enclave program through remote attestation, which essentially help the network establish trust on consensus participants.

Taking Hyperledger Sawtooth for example, the PoET protocol works in two phases for every participating node i:

1) *TEE setup*: Node i first obtains the PoET protocol program from a trusted source and instantiates it on its SGX machine wherein the random back-off routine runs inside an enclave. The trusted program generates a signing key pair $\langle PK_i, SK_i \rangle$ for node i and starts the attestation process that sends an attestation report to the network. The attestation report includes the public key PK_i and the enclave measurement signed by the Intel Enhanced Privacy Identification (EPID) private key inside the enclave. Other nodes in the network will validate the node i's hardware authenticity through Intel Attestation Service (IAS) and validate the attestation report before accepting node i.
2) *Participating in consensus*: The consensus process is similar to Nakamoto except for block generation and validation. For each block cycle, node i waits for a length of time dictated by the random back-off routine running in the enclave before producing a new block. The enclave then generates a certificate of back-off completion signed by node i's private key SK_i which is broadcast to the network along with the new block. Upon receipt of the new block, other nodes validate the block content and the back-off completion certificate with node i's public key PK_i. A block shall be appended to the blockchain if it passes validation and a certain finalization rule.

While PoET was initially proposed to substitute the PoW for block proposal without touching on the longest-chain rule, the usage of hardware-based public cryptography actually enables the use of more effcient BFT algorithms for block finalization. The hybrid PoET-BFT idea is currently used by Sawtooth PBFT, a sub-project of Hyperledger Sawtooth.

Security analysis PoET can tolerate up to 50% TEE nodes being malicious. Since every TEE node runs the same random back-off routine in the enclave, a player can shorten its expected back-off time by running multiple TEE nodes, which opens the door for Sybil attacks. If more than 50% of TEE devices collude or are controlled by an attacker, they can ultimately win the block race and keep extending the malicious chain. Therefore, PoET is most suited for permissioned blockchains, where every participant is authenticated and runs one TEE node.

For Hyperledger Sawtooth, Intel is the sole TEE hardware vendor and attestation service provider, which poses a single point of risk to the network. The validity of remote attestation depends on the integrity of SGX implementation and the availability of IAS. Recent attacks such as *cache attacks* [104], *Foreshadow* [105] and *Foreshadow-NG* [106] have demonstrated the ability to extract the EPID private key from hardware exploiting side channels and speculative execution, posing a security threat at the hardware level.

C. Proof of TEE-Stake (PoTS)

PoTS is another protocol harmonizing TEE and blockchain consensus. It was first proposed by Li et al. [107] in 2016 and was formalized by Andreina et al. [108] in 2019. A PoTS node i follows the same setup procedure as in PoET to bootstrap a TEE program, generate the signing key pair $\langle SK_i, PK_i \rangle$, and attest the setup to the network. Instead of simulating would-be elapsed time by PoW mining, the enclave program of PoTS is akin to Algorand's cryptographic sortition scheme that randomly selects a committee according to the stake ratios. Every node in the committee is eligible to propose a new block for the coming block cycle. To prove this eligibility to the network, a block also includes the eligibility proof signature $\sigma^{ep}_{SK_i}$, which is produced by the enclave program of the block generating node i. Once another node receive this block, they will validate the block content as well as the signature $\sigma^{ep}_{SK_i}$ using the corresponding public key PK_i. Then the longest-chain rule is used to determine whether to accept this block into the blockchain. Notably, the security offered by public key cryptography and TEE-certified execution of committee selection helps PoTS counter the stake-bleeding and stake-grinding attack.

Security analysis PoTS can tolerate up to 50% of all stake value at TEE nodes being maliciously controlled, the same as the fault tolerance of chain-based or committee based PoS. Compared to PoET, the incorporation of stakes gives PoTS higher robustness against Sybil attacks, which implies its applicability to permissionless blockchains. On the other hand, the single point of risk in TEE hardware vendor still exists.

D. Proof of Retrievability (PoR)

PoR was originally proposed by Juels et al. [109] in 2007 as a cryptographic building block for a semi-trusted distributed archiving system. The core feature of PoR is to allow a file owner to check if its online files/file fragments are securely stored and retrievable via a challenge-response protocol. The retrievability of a target file F at a remote node n_i can prove n_i indeed spends the required amount of storage resources to store F. Because of the space requirement behind retrievability, PoR is also known as proof of space.

In the role of a consensus protocol, PoR was first used by the cryptocurrency Permacoin, proposed by Miller et al. [110]

in 2014. It was designed as a mining-free alternative to PoW. First, a central dealer publishes a target dataset F and computes the digest of F (the merkle hash tree root of all segments of F). Then each participant stores some random segments of F per its storage capability, and computes the digest of these segments. For every block cycle, the dealer initiates a lottery game with a random puzzle. Then every participant derives a lottery ticket consisting of a fixed number of PoR challenges from its locally stored segments, public key, and the puzzle. Participants with more segments stored have higher probability of winning the lottery and thus get to generate a block. All PoR challenges are stored in the new block and verified by the whole network. Permacoin also implements a signature-based mechanism to discourage participants from outsourcing the storage task. Aside from PoR, Permacoin inherits Bitcoin for other consensus components.

Compared to PoW, PoR has two economical advantages. First, file storage in PoR consumes far less energy and storage space as a resource can be recycled. Second, PoR can be repurposed for meaningful storage tasks. For example the target dataset can be some extremely large but useful public dataset. In fact, the latter advantage is rarely seen in any other proof-of-X based protocol.

Security analysis Since the block winning rate of a participant is proportional to its local storage space, PoR can tolerate up to 50% of gross storage being held up by the malicious party. Although this still can trigger a bitter arm race of storage units, it downplays the efficacy of ASICs and encourages a wider variety of mining participants. On the other hand, the 50% threshold depends on the job of the central dealer to some extent. To ensure the diversity of lottery tickets across all participants and thus increase the randomness of the lottery, the target dataset should be large enough so that participants stores almost non-overlapping segments. This assumption can be undermined if the dealer chooses a not-large-enough dataset and poorly segments it.

E. RCPA

The Ripple consensus protocol/algorithm (RCPA) was proposed by Schwartz et al. [111] in 2014 as the underlying protocol for Ripple, a global payment and gross settlement network operated by the Ripple company. In comparison to public blockchains like Bitcoin and Ethereum, Ripple treats individual transactions as the ledger's atomic items, which is close to the original idea of distributed ledger system.

In Ripple network, only the server nodes can participate in consensus by collecting transactions from clients and proposing them to peer servers for consensus. In what follows, we will call server nodes just nodes for short. Specially, every node keeps a unique node list (UNL) which identifies the nodes it can trust and directly exchange messages with. A UNL relationship is reciprocal. We call a group of nodes that are fully connected by UNL relationships a UNL clique.

RCPA runs in rounds each of which finalizes a certain set of transactions into the ledger. The operation of RCPA at one node is shown in Algorithm 8. Specially, the yes-vote threshold of the final round $TH_{MaxRound} = 80\%$, which is designed to cope with the network connectivity assumption that any two UNL cliques UNL_i, UNL_j must have at least $\frac{1}{5}\max(|UNL_i|, |UNL_j|)$ inter-clique UNL relationships. In an ideal environment, as long as $MaxRound$ is large enough, all valid transactions sent from non-faulty nodes should eventually surpass the 80% yes-vote threshold.

Algorithm 8: RPCA (validator node)

Joining Ripple network as a validator;
`/* Main loop */`
for *new block cycle* **do**
  Collect valid transactions (new or leftovers from previous cycles) $\Rightarrow CandidateSet$;
  for $r = 1 \rightarrow MaxRound$ **do**
    Broadcast $CandidateSet$ to UNL peers;
    After receiving transactions from UNL peers, add them to $CandidateSet$ and broadcast a vote on the veracity (yes/no) of every transaction;
    After receiving votes from UNL peers, discard transactions from $CandidateSet$ whose yes-votes fall short of a threshold TH_r ($TH_{MaxRound} = 80\%$);
  end
  The remaining transactions in $CandidateSet$ are admitted into the new block;
end

Security analysis RCPA is essentially a relaxed DLS protocol with a BFT bound of 1/5. It further requires no more than 1/5 of nodes are faulty in every UNL clique in order to ensure overall network consensus. Compared to PBFT that achieves 1/3 Byzantine fault tolerance with $O(N^2)$ message complexity in a fully connected network, RPCA's 1/5 fault tolerance bound trades for a lower connectivity requirement and thus lower message complexity per block cycle, which is $O(MK^2) = O(NK)$ where K is the clique size and $M = N/K$ is the number of cliques. Therefore Ripple is a good example of trading fault tolerance for better performance when a certain level of trust is assumed.

Ripple's current major customers are established corporations and financial institutions. Part of the reason is that the 1/5 fault tolerance can be too restrictive for low-trust environments. Interestingly, Stellar [112], originally a fork project from the Ripple, has shifted to PBFT-style consensus and achieves the 1/3 fundamental fault tolerance bound.

F. IOTA Tangle

IOTA is a blockchain initiative designed for machine-to-machine micro payments in the IoT setting. Tangle, its underlying consensus protocol, was formalized by Popov [113] in 2016. The key innovation of Tangle is its ledger data structure: a directed acyclic graph (DAG) instead of a single chain of blocks. Notably, every vertex of the DAG represents a single transaction and diverging branches must hold different transactions. This contrasts with the blockchain norm that a block may contain multiple transactions, which is impractical

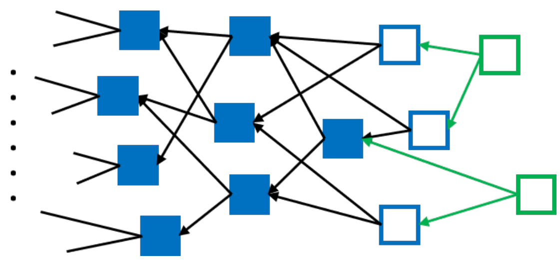


Fig. 10. An example of IOTA Tangle's DAG structure. Every new transaction (hollow green) needs to approve two unapproved transactions ("tips", hollow blue) and becomes a new tip. If only one tip is available, choose an approved transaction for the other. The arrow direction symbolizes a user's tip selection behavior.

to Tangle's DAG ledger structure as diverging blocks very likely contain some identical transactions.

IOTA Tangle does not reward block miners with cryptocurrency. Instead, to incentivize nodes' participation in validation, every user needs to approve (i.e. validate) two unapproved transactions or "tips" of the DAG in order to submit a new transaction. If there is only one tip available, the user chooses an approved transaction instead. The user then attaches the hashes of the two chosen transactions to the new transaction, and solves a PoW puzzle before broadcasting it. As a result, every vertex in the DAG has an out-degree of two, as is shown in Figure 10.

When multiple conflicting tips (i.e. double-spending) are detected by a user, only one can be deemed valid and approved. In this case the user resorts to a tip-selection scheme to choose one that yields the highest acceptance probability in the long term. Other conflicting tips are considered invalid and then orphaned. Tangle currently employs a Markov-chain Monte Carlo (MCMC) based scheme to simulate the acceptance probabilities of each of the conflicting tips.

When there are more than two non-conflicting tips available (i.e. multiple valid choices), the user chooses two tips according to a weight based strategy, which works as follows. Every transaction is assigned an initial weight, which is proportional (discrete values) to the PoW effort spent on this transaction. The user chooses two tips with the highest weights. If the user only possesses a subgraph of the DAG, it performs two weighted random walks from the beginning of the subgraph to decide two tip choices [114]. The weights of a tip transaction and its preceding transactions along the lineage of approvals in the DAG accumulate by the weight of the approving transaction. As a result, as time goes by the DAG ledger will look like a cascade of transactions that advances toward the direction of the highest accumulative weight. The MCMC-based scheme, the highest-accumulated-weight strategy, and the weighted random walk algorithm can be regarded as the DAG version of longest-chain rule, and essentially contribute to Tangle's tamper-resistance and probabilistic finality.

Theoretically, as is claimed in IOTA official website, Tangle's DAG data structure can boost IOTA's transaction capacity to millions per second (or unbounded). This is due to the DAG's native support for branches: when multiple valid tips are present, a new transaction just chooses two to approve, while leaving the remaining tips to other new transactions. As a result, the more new transactions come out, the more likely a tip gets approved. Therefore, Tangle is said to scale well with both network size and transaction rate, perfectly suiting the IoT machine-to-machine micro-payment setting. In practice, its transaction capacity is still capped by link-/physical-layer bandwidth.

Security analysis Since PoW is enforced for transaction generation, theoretically Tangle's probabilistic finality is guaranteed as long as the honest nodes control more than 50% of gross computing power. Nonetheless, the DAG-structured ledger along with its current version of MCMC-based tip-selection is susceptible to parasite chain attack, as is documented in [113]. An attacker can grow a parallel chain from some early point all the way to the current DAG height, with intermittent connections to the DAG. This takes the attacker significantly less than 50% of network computing power, as the malicious chain may carry far fewer transactions than the main DAG. To address this issue, Tangle currently relies on a centralized checkpointing scheme to periodically confirm past transactions. The IOTA community plans to improve the MCMC-based tip-selection scheme in hope of obtaining the 50% fault tolerance in a truly decentralized fashion.

Moreover, the assumption that more than 50% of computing power are honest is questionable in a practical IoT setting. Because of the heterogeneous and power-saving nature of IoT devices, it is possible that a well-financed, continuously-powered adversary can leverage more computing power than the entire IoT device network. It is also questionable whether a permissionless blockchain is suitable for IoT applications.

Other DAG-based consensus protocols include SPECTRE [115], PHANTOM [116], Conflux [117], and GraphChain [118]. Generally, protocols using non-linear ledger structures such as DAG marks a significant divergence from Nakamoto's blockchain design. Their key insight is that transaction capacity should not be limited by a restrictive consensus object, such as a linearly growing chain of blocks with fixed intervals. Instead, the influx of transactions should be the driver of ledger expansion. Nonetheless, the security impact of a less restrictive ledger can be profound, exampled by the parasite chain attack against IOTA.

VII. Comparison and Discussion

A. Comparing Different Protocols

Every consensus protocol has its unique design philosophy and purpose. Table II compares the protocols that we have discussed earlier from three aspects: the five-component framework composition, fault tolerance, and transaction capacity. For some consensus protocols, not all of the five components are specified in the white paper. For example, protocols designed for permissioned blockchains (PoET, PoTS) don't need an incentive mechanism; protocols that were initially proposed to substitute PoW (PoA, PoET, PoR) assume Bitcoin's framework by default; protocols that were designed from the classical distributed consensus (PBFT, HoneybadgerBFT) only need to specify information propagation and finalization.

The fault tolerance capability largely depends on the block finalization mechanism used. For example, the 50% bound

TABLE II
A Comparison of Blockchain Consensus Protocols

Consensus Protocol (Blockchain Realizations)	**Five-Component Framework**					**Fault Tolerance**	**Transaction Capacity** (tx/sec)
	Block Proposal	**Block Validation***	**Information Propagation**	**Block Finalization**	**Incentive Mechanism**		
Nakamoto (Bitcoin, Litecoin)	PoW	PoW check	Gossiping	Longest-chain rule†	Block reward and transaction fee	50% computing power	Sub-ten
Nakamoto (Ethereum)	PoW (Ethash)	PoW check	Gossiping via secure channels	A variation of GHOST rule†	Block reward and transaction fee	50% computing power	Tens
Bitcoin-NG (Waves-NG)	PoW for key blocks	PoW check for key blocks	Gossiping	Longest-chain rule†	Block reward and transaction fee	50% computing power	Hundreds
Hybrid PoW-BFT (PeerConsens, SCP, ByzCoin)	PoW (blockchain decides the BFT committee)	PoW check	Broadcast among BFT committee	Longest-chain rule† for chain (PBFT for transactions)	⊙	33% computing power	Hundreds
Chain-based PoS (Peercoin, Nxt, PoAct)	PoS	PoS check	Gossiping	Longest-chain rule†	Block reward and transaction fee	50% deposited stake value	Tens
Committee-based PoS (Ouroboros, Praos, CoA, Snow White)	PoS-based committee election	Proposer eligibility check	Broadcast among committee	Longest-chain rule†	Block reward	50% token wealth	Hundreds
BFT-based PoS (Tendermint)	PoS-based round robin	Proposer eligibility check	Broadcast among validators	BFT (adapted DLS)	Block reward	33% token wealth	Thousands
BFT-based PoS (Algorand)	PoS-based committee election	Proposer eligibility check	Broadcast among committee	BFT (adapted Byzantine agreement)	Block reward	33% token wealth	Thousands
BFT-based PoS (Casper FFG)	PoW (Ethash)	PoW & Checkpoint tree check	Broadcast among validators	BFT (with staked votes)	Block reward for miners and validators	33% deposited stake value	(Unavailable)
DPoS (EOS, Lisk, BitShares)	PoS with stake delegation	Delegate eligibility check	Broadcast among delegates	BFT (suggested, not limited to)	Block reward	33% delegates	Thousands
PoA (Rinkby, Kovan, POA Network)	PoA	Block proposer identity check	⊙ Broadcast◇	⊙ HoneyBadger-BFT◇	⊙ Transaction fee◇	(Depends on finalization)	Tens
PoET (Hyperledger Sawtooth family)	PoET within TEE	TEE certificate check	⊙ Broadcast◇	⊙ PBFT◇	⊙	(Depends on finalization)	Thousands if PBFT is used
PoTS	PoTS-based committe election	Proposer eligibility&TEE cert. check	⊙	⊙	⊙	50% stake value in TEEs	(Unavailable)
PoR (Permacoin)	PoR	File retrievability check	⊙	⊙	⊙	50% storage space	(Unavailable)
RPCA (Ripple)	Any server can propose transactions	UNL membership check	Broadcast to UNL peers	Accepting $> 80\%$ voted transactions	Transaction fee	20% nodes in each UNL	Thousands
Tangle (IOTA)	Approving 2 tips & PoW	PoW & tip approval check	Gossiping	Highest cumulative weight rule†	Eligibility for issuing new transactions	50% computing power	Thousands, potentially millions
PBFT (Hyperledger Fabric, Stellar)	⊘	⊘	Broadcast	Partially sync. BFT for transactions	⊘ Transaction fee◇	33% servers	Thousands
HoneyBadger-BFT (POA Network)	⊘	⊘	Broadcast	Async. BFT for transactions	⊘ Transaction fee◇	33% servers	Thousands

*Block validation also includes validating all transactions inside the block, which is omitted here for space saving.
†: With probabilistic finality. ⊘: Unspecified in the protocol white paper. ⊙: Unspecified in the protocol white paper, but the Bitcoin counterpart is usable. ◇: Found in one or more blockchain realizations.

(of computing power or stake value) applies to all protocols that inherit the probabilistic finality property from Bitcoin; protocols using a BFT scheme for block finalization typically achieve 33% fault tolerance and guarantee deterministic finality.

The transaction capacity statistics come from either the protocol white papers or the official documentations of corresponding blockchain implementation, and only specify the scales of magnitude, i.e. sub-ten, tens, hundreds, or thousands. Notably, according the simulation study in various white papers for BFT-based protocols [16], [112], [15], for BFT to process thousands of transactions per second the network size should be small, typically around several hundred.

B. Discussion on Consensus Design Philosophy

The five-component framework demonstrates the composability of a blockchain consensus protocol, which implies the feasibility of harmonizing different consensus protocol ideas based on different application needs. For example, Sawtooth-PBFT, a member of the Hyperledger Sawtooth family that is specially designed for permissioned blockchains, harmonizes PoET with PBFT, with block proposal utilizing the former while block finalization adapted from the latter. Similarly, the POA Network uses PoA for block proposal while HoneyBadgerBFT for block finalization for best performance under asynchronous network conditions. These examples indicate that we can leverage the five-component framework for consensus protocol design and innovate an individual component while inheriting the others from established protocols. However, there is never a one-for-all blockchain consensus protocol; specific application scenarios and goals should be considered.

The trilemma between decentralization, security, and scalability Depending on the application scenario, a desired protocol needs to strike a balance between three metrics: decentralization, security, and scalability (scaling with transaction throughput and network size). For most blockchains especially those designed for financial applications, security is always the top priority; meanwhile the compromise between decentralization and scalability depends on the practical needs. For example, as Bitcoin operates in a zero-trust environment where no authentication of any kind is enforced, Nakamoto consensus protocol needs to rely on decentralized best-effort mining to achieve the security against double spending, at the price of low transaction throughput. In comparison, BFT-based protocols are designed for more benign environments where every participant operates with revealed identity and the overall network is well connected. This reduction of decentralization allows for a much higher transaction throughput but lower scalability with network size.

In addition, a consensus protocol designer should also anticipate the dynamics between a protocol and the blockchain network's decentralization profile. For instance, for protocols using the gossiping rule and longest-chain rule, the heterogeneity of network connection allows well-connected nodes to disseminate new blocks faster and more thoroughly than the less-connected ones, effectively enhancing the former's block winning chance. For financially incentivized blockchain systems such as Bitcoin and Ethereum, the centralization of mining power and token wealth leads to a greater risk of monopoly. All these issues require a careful examination on each of the five protocol components while taking into account the real circumstances (network connectivity, trust level, etc.) and application need.

VIII. Conclusion

In this survey we provided a succinct summary of classical fault-tolerance consensus research, a five-component framework for a general blockchain consensus protocol, and a comprehensive review of blockchain consensus protocols that have gained great popularity and potential. We also qualitatively analyzed these protocols and highlighted their fault tolerance, performance, and potential vulnerabilities. Notably, many of these protocols are still under development and are subject to major changes at the time of writing.

We hope the five-component framework, classification methodology as well as protocol abstractions covered in this survey can help researchers and developers grasp the fundamentals of blockchain consensus protocols and facilitate future designs.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] "Global bitcoin nodes distribution," Bitnodes. [Online]. Available: https://bitnodes.earn.com/

[3] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, "On scaling decentralized blockchains," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 106–125.

[4] "Visa fact sheet," VISA Inc. [Online]. Available: https://usa.visa.com/dam/VCOM/download/corporate/media/visanet-technology/aboutvisafactsheet.pdf

[5] "Bitcoin energy consumption index," Digiconomist. [Online]. Available: https://digiconomist.net/bitcoin-energy-consumption

[6] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," *self-published paper, August*, vol. 19, 2012.

[7] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol." in *NSDI*, 2016, pp. 45–59.

[8] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[9] P. Daian, R. Pass, and E. Shi, "Snow white: Robustly reconfigurable consensus and applications to provably secure proofs of stake," Technical Report. Cryptology ePrint Archive, Report 2016/919, Tech. Rep., 2017.

[10] "EOS.IO technical white paper v2," EOS.IO, 2018. [Online]. Available: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md

[11] "POA network whitepaper," POA Network, 2018. [Online]. Available: https://github.com/poanetwork/wiki/wiki/POA-Network-Whitepaper

[12] J. Kwon, "Tendermint: Consensus without mining," *Retrieved May*, vol. 18, p. 2017, 2014.

[13] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 51–68.

[14] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017.

[15] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," *arXiv preprint arXiv:1801.10228*, 2018.

[16] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 31–42.

[17] S. Duan, M. K. Reiter, and H. Zhang, "Beat: Asynchronous bft made practical," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 2028–2041.
[18] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *International Workshop on Open Problems in Network Security*. Springer, 2015, pp. 112–125.
[19] C. Cachin and M. Vukolić, "Blockchains consensus protocols in the wild," *arXiv preprint arXiv:1707.01873*, 2017.
[20] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis, "Consensus in the age of blockchains," *arXiv preprint arXiv:1711.03936*, 2017.
[21] W. Wang, D. T. Hoang, Z. Xiong, D. Niyato, P. Wang, P. Hu, and Y. Wen, "A survey on consensus mechanisms and mining management in blockchain networks," *arXiv preprint arXiv:1805.02707*, 2018.
[22] Y. Xiao, N. Zhang, J. Li, W. Lou, and Y. T. Hou, "Distributed consensus protocols and algorithms," *Blockchain for Distributed Systems Security*, p. 25, 2019.
[23] H. Attiya, A. Bar-Noy, and D. Dolev, "Sharing memory robustly in message-passing systems," *Journal of the ACM (JACM)*, vol. 42, no. 1, pp. 124–142, 1995.
[24] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.
[25] H. Attiya and J. Welch, *Distributed computing: fundamentals, simulations, and advanced topics*. John Wiley & Sons, 2004, vol. 19.
[26] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.
[27] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.
[28] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.
[29] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM (JACM)*, vol. 27, no. 2, pp. 228–234, 1980.
[30] B. Alpern and F. B. Schneider, "Defining liveness," *Information processing letters*, vol. 21, no. 4, pp. 181–185, 1985.
[31] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.
[32] ——, "Using time instead of timeout for fault-tolerant distributed systems," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 6, no. 2, pp. 254–280, 1984.
[33] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys (CSUR)*, vol. 22, no. 4, pp. 299–319, 1990.
[34] F. Cristian, H. Aghili, R. Strong, and D. Dolev, *Atomic broadcast: From simple message diffusion to Byzantine agreement*. Citeseer, 1986.
[35] G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," *Journal of the ACM (JACM)*, vol. 32, no. 4, pp. 824–840, 1985.
[36] X. Défago, A. Schiper, and P. Urbán, "Total order broadcast and multicast algorithms: Taxonomy and survey," *ACM Computing Surveys (CSUR)*, vol. 36, no. 4, pp. 372–421, 2004.
[37] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM (JACM)*, vol. 43, no. 2, pp. 225–267, 1996.
[38] B. M. Oki and B. H. Liskov, "Viewstamped replication: A new primary copy method to support highly-available distributed systems," in *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*. ACM, 1988, pp. 8–17.
[39] B. Liskov and J. Cowling, "Viewstamped replication revisited," 2012.
[40] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems (TOCS)*, vol. 16, no. 2, pp. 133–169, 1998.
[41] L. Lamport *et al.*, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.
[42] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, 2014, pp. 305–319.
[43] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie, "Fault-scalable byzantine fault-tolerant services," *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5, pp. 59–74, 2005.
[44] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira, "Hq replication: A hybrid quorum protocol for byzantine fault tolerance," in *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 177–190.
[45] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: speculative byzantine fault tolerance," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 45–58, 2007.
[46] J.-P. Martin and L. Alvisi, "Fast byzantine consensus," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 3, pp. 202–215, 2006.
[47] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung, "Spin one's wheels? byzantine fault tolerance with a spinning primary," in *Reliable Distributed Systems, 2009. SRDS'09. 28th IEEE International Symposium on*. IEEE, 2009, pp. 135–144.
[48] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making byzantine fault tolerant systems tolerate byzantine faults." in *NSDI*, vol. 9, 2009, pp. 153–168.
[49] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić, "The next 700 bft protocols," in *Proceedings of the 5th European conference on Computer systems*. ACM, 2010, pp. 363–376.
[50] A. N. Bessani, "(BFT) state machine replication: the hype, the virtue, and even some practice," Tutorials in EuroSys 2012. [Online]. Available: http://www.di.fc.ul.pt/~bessani/publications/T1-bftsmr.pdf
[51] G. Bracha, "Asynchronous byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.
[52] M. Ben-Or, "Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols," in *Proceedings of the second annual ACM symposium on Principles of distributed computing*. ACM, 1983, pp. 27–30.
[53] M. O. Rabin, "Randomized byzantine generals," in *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*. IEEE, 1983, pp. 403–409.
[54] G. Bracha, "An asynchronous [(n-1)/3]-resilient consensus protocol," in *Proceedings of the third annual ACM symposium on Principles of distributed computing*. ACM, 1984, pp. 154–162.
[55] M. Ben-Or, B. Kelmer, and T. Rabin, "Asynchronous secure computations with optimal resilience," in *Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing*. ACM, 1994, pp. 183–192.
[56] R. Canetti and T. Rabin, "Fast asynchronous byzantine agreement with optimal resilience," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. ACM, 1993, pp. 42–51.
[57] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Annual International Cryptology Conference*. Springer, 2001, pp. 524–541.
[58] C. Cachin and S. Tessaro, "Asynchronous verifiable information dispersal," in *Reliable Distributed Systems, 2005. SRDS 2005. 24th IEEE Symposium on*. IEEE, 2005, pp. 191–201.
[59] A. Mostéfaoui, H. Moumen, and M. Raynal, "Signature-free asynchronous byzantine consensus with $t < n/3$ and $o(n^2)$ messages," in *Proceedings of the 2014 ACM symposium on Principles of distributed computing*. ACM, 2014, pp. 2–9.
[60] J. Baek and Y. Zheng, "Simple and efficient threshold cryptosystem from the gap diffie-hellman group," in *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, vol. 3. IEEE, 2003, pp. 1491–1495.
[61] C. Decker, J. Seidel, and R. Wattenhofer, "Bitcoin meets strong consistency," in *Proceedings of the 17th International Conference on Distributed Computing and Networking*. ACM, 2016, p. 13.
[62] "EIP 1011: Hybrid Casper FFG," Ethereum Improvement Proposals (EIPs). [Online]. Available: https://eips.ethereum.org/EIPS/eip-1011
[63] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.
[64] "Litecoin official website," Litecoin Foundation. [Online]. Available: https://litecoin.org/
[65] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*. IEEE, 2013, pp. 1–10.
[66] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 281–310.
[67] "Visanet: The technology behind visa," Visa Inc. [Online]. Available: https://usa.visa.com/dam/VCOM/download/corporate/media/visanet-technology/visa-net-booklet.pdf
[68] "Bitcoin energy consumption index," Digiconomist. [Online]. Available: https://digiconomist.net/bitcoin-energy-consumption
[69] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network." in *USENIX Security Symposium*, 2015, pp. 129–144.

[70] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 3–16.

[71] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *International conference on financial cryptography and data security*. Springer, 2014, pp. 436–454.

[72] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 515–532.

[73] D. Gutteridge, "Japanese cryptocurrency monacoin hit by selfish mining attack," 2018. [Online]. Available: https://www.ccn.com/japanese-cryptocurrency-monacoin-hit-by-selfish-mining-attack/

[74] A. E. Gencer, S. Basu, I. Eyal, R. Van Renesse, and E. G. Sirer, "Decentralization in bitcoin and ethereum networks," *arXiv preprint arXiv:1801.03998*, 2018.

[75] D. Kondor, M. Pósfai, I. Csabai, and G. Vattay, "Do the rich get richer? an empirical analysis of the bitcoin transaction network," *PloS one*, vol. 9, no. 2, p. e86197, 2014.

[76] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 507–527.

[77] "Wave-ng protocol," Wave Platform. [Online]. Available: https://docs.wavesplatform.com/en/waves-environment/waves-protocol/waves-ng-protocol.html

[78] M. K. Reiter, "A secure group membership protocol," *IEEE Transactions on Software Engineering*, vol. 22, no. 1, pp. 31–42, 1996.

[79] L. Luu, V. Narayanan, K. Baweja, C. Zheng, S. Gilbert, and P. Saxena, "Scp: A computationally-scalable byzantine consensus protocol for blockchains," *See https://www. weusecoins. com/assets/pdf/library/SCP*, vol. 20, no. 20, p. 2016, 2015.

[80] "Sharding FAQs: On sharding blockchains," Ethereum Foundation. [Online]. Available: https://github.com/ethereum/wiki/wiki/Sharding-FAQ

[81] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 279–296.

[82] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping authorities" honest or bust" with decentralized witness cosigning," in *2016 IEEE Symposium on Security and Privacy (SP)*. Ieee, 2016, pp. 526–545.

[83] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," in *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[84] ——, "Fruitchains: A fair blockchain," in *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 2017, pp. 315–324.

[85] A. Poelstra *et al.*, "Distributed consensus from proof of stake is impossible," *Self-published Paper*, 2014.

[86] "Nxt whitepaper revision 4," Nxt community, 2014. [Online]. Available: https://www.dropbox.com/s/cbuwrorf672c0yy/NxtWhitepaper_v122_rev4.pdf

[87] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, "proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract] y," *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 3, pp. 34–37, 2014.

[88] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," in *Proceedings of the twenty-first annual ACM symposium on Theory of computing*. ACM, 1989, pp. 73–85.

[89] I. Bentov, A. Gabizon, and A. Mizrahi, "Cryptocurrencies without proof of work," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 142–157.

[90] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 66–98.

[91] R. Pass and E. Shi, "The sleepy model of consensus," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 380–409.

[92] "Bitshares 2.0 documentation," The BitShares Organization, 2015. [Online]. Available: http://docs.bitshares.org/bitshares/index.html

[93] "Liskś consensus algorithm," Lisk. [Online]. Available: https://lisk.io/documentation/lisk-protocol/consensus

[94] D. Larimer, "DPOS BFT pipelined byzantine fault tolerance," 2018. [Online]. Available: https://medium.com/eosio/dpos-bft-pipelined-byzantine-fault-tolerance-8a0634a270ba

[95] J. Martinez, "Understanding proof of stake: The nothing at stake theory," 2018. [Online]. Available: https://medium.com/coinmonks/understanding-proof-of-stake-the-nothing-at-stake-theory-1f0d71bc027

[96] M. Franklin, "A survey of key evolving cryptosystems," *International Journal of Security and Networks*, vol. 1, no. 1-2, pp. 46–53, 2006.

[97] V. Buterin, Ethereum Foundation, 2014. [Online]. Available: https://blog.ethereum.org/2014/05/15/long-range-attacks-the-serious-problem-with-adaptive-proof-of-work/

[98] P. Gaži, A. Kiayias, and A. Russell, "Stake-bleeding attacks on proof-of-stake blockchains," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, 2018, pp. 85–92.

[99] "Grinding attack on proof of stake," The Ethereum Foundation. [Online]. Available: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs#how-does-validator-selection-work-and-what-is-stake-grinding

[100] "Aura - authority round - wiki," Parity Tech Documentation. [Online]. Available: https://wiki.parity.io/Aura

[101] "Hyperledger sawtooth project," The Linux Foundation, 2018. [Online]. Available: https://www.hyperledger.org/projects/sawtooth

[102] V. Costan and S. Devadas, "Intel SGX explained." *IACR Cryptology ePrint Archive*, vol. 2016, p. 86, 2016.

[103] "Introducing arm trustzone," Arm Ltd, 2018. [Online]. Available: https://developer.arm.com/technologies/trustzone

[104] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure:{SGX} cache attacks are practical," in *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*, 2017.

[105] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 991–1008.

[106] O. Weisse, J. Van Bulck, M. Minkin, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, R. Strackx, T. F. Wenisch, and Y. Yarom, "Foreshadow-ng: Breaking the virtual memory abstraction with transient out-of-order execution," Technical report, Tech. Rep., 2018.

[107] W. Li, S. Andreina, J.-M. Bohli, and G. Karame, "Securing proof-of-stake blockchain protocols," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2017, pp. 297–315.

[108] S. Andreina, J.-M. Bohli, W. Li, G. O. Karame, and G. A. Marson, "Potsa secure proof of tee-stake for permissionless blockchains."

[109] A. Juels and B. S. Kaliski Jr, "Pors: Proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security*. Acm, 2007, pp. 584–597.

[110] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *2014 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2014, pp. 475–490.

[111] D. Schwartz, N. Youngs, A. Britto *et al.*, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, vol. 5, 2014.

[112] D. Mazieres, "The stellar consensus protocol: A federated model for internet-level consensus," *Stellar Development Foundation*, 2015.

[113] S. Popov, "The tangle," 2016. [Online]. Available: http://www.descryptions.com/Iota.pdf

[114] "Tangle tip selection," IOTA Foundation. [Online]. Available: https://docs.iota.org/docs/the-tangle/0.1/concepts/tip-selection#in-depth-explanation-of-the-tip-selection-algorithm

[115] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "Spectre: A fast and scalable cryptocurrency protocol." *IACR Cryptology ePrint Archive*, vol. 2016, p. 1159, 2016.

[116] Y. Sompolinsky and A. Zohar, "Phantom: A scalable blockdag protocol." *IACR Cryptology ePrint Archive*, vol. 2018, p. 104, 2018.

[117] C. Li, P. Li, W. Xu, F. Long, and A. C.-c. Yao, "Scaling nakamoto consensus to thousands of transactions per second," *arXiv preprint arXiv:1805.03870*, 2018.

[118] X. Boyen, C. Carr, and T. Haines, "Graphchain: A blockchain-free scalable decentralised ledger," in *Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts*. ACM, 2018, pp. 21–33.