SoK: MetaAnalysis of Alternative Consensus Protocols for Blockchains

Abstract—**Blockchain technology has been gaining popularity in the public, in industry, and in academia. However, these systems are currently limited in their ability to support robust, large-scale applications due to scalability and security challenges. A fundamental step to resolving some of these challenges lies in the consensus layer. As such, both industry and academia are heavily investing in alternative consensus research and development, seeking alternatives to Bitcoin's Nakamoto consensus. Many of these protocols favor Proof of Stake (PoS) as a Sybil control mechanism to today's Proof of Work (PoW). However, thus far there has been no comprehensive, rigorous comparison of these alternative consensus protocols. This paper presents a systematization of knowledge within these major blockchain protocols, understanding the common challenges and solutions, and providing a formal structure within which to compare them. We break down these protocols by network, adversarial, and economic model, deeply understanding the common challenges of choosing proposers and committees, propagation, and finality.**

1. Introduction

Blockchain technology presents a groundbreaking innovation, calling into question long established paradigms in distributed systems computing and financial theory, and providing a novel platform for decentralized internet applications. In particular, cryptocurrencies (like Bitcoin and Ethereum) have gained attention for some distinct benefits they offer over fiat currency including increased transparency and resistance to counterfeiting among other parameters [8]. The transactions of these currencies are backed by distributed ledgers that are implemented by data structures known as blockchains. *Blockchain* here refers to "a decentralized, replicated, immutable and tamper-evident log" [7]. Crucially, "data on the blockchain cannot be deleted, and anyone can read data from the blockchain and verify its correctness" [7]. Blockchains are composed of *blocks*, which are structures containing transactions. Further, a vital aspect of blockchains, at least in Nakaomto's initial instantiation, is *permissionless consensus*, where "anyone can join (or leave) the protocol execution (without getting permission from a centralized or distributed authority), and authentication mechanisms are not available. Additionally, participants may join and leave the system "at will" [30].

The computers that secure these blockchains and add new transactions to the ledger are known as *miners* [1]. These miners have a monetary incentive to come to consensus with other miners regarding the ledger's history by mining on the "longest chain". This process is referred to as *Nakamoto Consensus*, which is backed by a process known as *Proof of Work* (PoW) [7].

1.1. Problem Definition

In PoW Nakamoto based consensus, PoW is wasteful of energy and computational power, and Nakamoto consensus favors availability over consistency. [12]. As such, alternative consensus protocols have been proposed, many favoring *Proof of Stake* (PoS) as a Sybil control mechanism and providing varying options for finality gaurantees, encompassing both availability favoring and consistency favoring protocols. In a PoS system, consensus is reached through the agreement of validators who have bonded their coins to the network. These validators are akin to miners in PoW. However, in PoS, validators create a deposit (a bond) of coins which allows them to propose and vote on new blocks. While PoW in Bitcoin functions as both a Sybil control mechanism, and as a source of randomness to select a block maker, PoS only serves as a Sybil control mechanism and thus requires a separate randomness protocol for choosing proposers. Proponents argue that PoS resolves challenges with PoW such as energy inefficiency, high latency, and centralization risk, to name a few [15].

To date, there does not exist a sufficient comprehensive analysis of alternative consensus protocols favoring PoS. This category of protocols is being deeply explored by major research groups and projects in the space, and has the potential to power more scalable and energy efficient blockchain platforms. Building consensus protocols is fundamentally non-trivial, and requires a deep understanding of previous work to most efficiently build upon current solutions or adopt existing solutions to a particular use case. Our goal is to provide a comprehensive analysis of major alternative blockchain consensus protocols, evaluating each within a common framework to allow for accurate and efficient comparison. Here, we classify protocols by network, adversarial and economic model, and understand the common challenges in proposer and committee election, message propagation, finalization, handling churn, security of randomness generation, network partition resolution, and scalability. In particular, we focus our analysis on Tendermint [37], Thunderella [36], Algorand [4], Dfinity [13], Ouroboros Genesis [25], Casper FFG [39], and Casper TFG [40].

1.2. Related Works

This work follows an academic succession of literature providing systematic reviews and meta analyses of Bitcoin and related topics. Bonneau et. al [9] perform a mathematical evaluation of cryptocurrencies. Zohar [34] examines scalability issues pertaining to blockchains, especially Bitcoin. Pass and Shi [30] mathematically analyze blockchain-based permissionless, sleepy consensus, demonstrating that it is strictly harder than classical consensus. Croman et. al [12] present a position paper on scalability bottlenecks and proposed solutions.

Indeed, there have been academic works encouraging the shift from PoW to PoS and calling for more robust analysis frameworks, including Vukolic [33], which encourages use of classical BFT protocols over PoW for reasons of safety and fork-resistance. Bano et al. [7] investigate blockchain protocols for consensus, providing a survey of different types of alternative consensus protocols that aim to be used in permissionless environments; and Gervais et al. [19] describe a framework evaluating the security and performance of PoW blockchains.

1.3. Problem Definition

1.3.1. Consensus in the Classical Setting. The consensus problem is the issue of multiple processors coming to agreement on an output. [5] To solve the consensus problem, termination, agreement, and validity must be guaranteed. *Termination* means that every non faulty node is assigned some value. *Agreement* refers to all correct nodes terminating on the same value. *Validity* means that if all nodes propose the same value then all correct nodes must decide on that particular value.

1.3.2. Consensus in the Blockchain Setting. Notably, blockchain consensus presents a departure from traditional consensus. Blockchains are meant to handle significant churn, and are typically meant to deal with permissionless consensus [30].

These protocols have certain desired properties, which we define here. Pass and Shi note *consistency* as a prerequisite for consensus, which requires that ”all honest nodes logs agree with each other although some nodes may progress faster than others” [30]. This can be thought of as similar to agreement. Additionally, Pass and Shi describe a required *liveness* property by which ”transactions submitted by an honest user get incorporated into the ledger sufficiently fast” [17]. This can be considered as similar to the termination property. A *safety* property, explicitly specified as desirable quality in many of the aforementioned works, is defined simply as ”one which states that something [undesirable] will not happen” [22]. So, when referring to safety, these protocols specify which unwanted protocol behavior they avoid. *Availability* requires that ”every request received by a non-failing node in the system must result in a response” [20]. From this, Brewer’s CAP theorem states that in the presence of a network partition, a protocol cannot guarantee both availability and consistency, a fundamental tradeoff each of these protocols is forced to make [20].

Moreover, the following are the distinct parties present in the blockchain setting. *Nodes* are any actors who interact with the blockchain and *validators* are those who have staked by bonding currency in a PoS system. There exists a *validator-set* comprised of all validators in a system, and a *committee*, which is the set of validators chosen to verify blocks at some point in time in a system. A *proposer* is a validator chosen from the validator set, who collects transactions and forms blocks.

There are two major categories of PoS protocols: chain-based and PBFT-style [15]. In *chain-based* systems, a validator is pseudo-randomly chosen to be a block *proposer* to create a block which is then extended by future proposers in the rounds that follow. This is meant to simulate Nakamoto PoW consensus. In this case, blocks are probabilistically finalized as they move deeper into the chain. In *BFT-style* PoS blockchains, one of these validators is randomly chosen as a proposer to propose a new block to add to the blockchain in every round. The rest of the validators vote on the inclusion of this new block, with voting power in proportion to the bonds they have staked. Consensus is achieved when some pre-specified portion of validators agree on accepting the new block.

1.4. Model

1.4.1. Network and Computational Synchrony Model. In both traditional distributed systems literature and blockchain consensus, we consider the message passing model in which processors communicate by sending messages between bidirectional communication channels [5]. In blockchain consensus protocols, these nodes form a peer-to-peer network, meaning that every message is propagated to every node in the network via a gossip protocol. Different protocols adopt different timing models based on the environment in which they are designed to function. That is, some protocols are designed to work in unreliable networks that drop messages and may cause arbitrary message delay, like the Internet, whereas other protocols are optimized for extremely reliable channels, like permissioned company intranets. These protocols are said to be operating under differing assumptions of synchrony [23].

In a *fully synchronous* network, there is a known upper bound on message delay and all messages are received in the exact linear ordering in which they were sent. This assumption is often considered to be unrealistic in practice since it assumes the existence of a global synchronization clock which is practically infeasible in a large, permissionless distributed system [23]. Full synchrony allows algorithms to operate in rounds, since this upper bound on message transmission exists. *Synchronous* networks assume a known upper bound on message delay, however messages do not need to be ordered [23].

Semi-synchrony also assumes the existence of an upper bound not known a priori, however the distribution of the upper bound is known [6].

TABLE 1. MODEL

	Tendermint	Thunderella	Algorand	Dfinity	Ouroboros G.	Casper FFG	Casper TFG
Network Model	Partial Sync.	Sync.	Partial Sync.	Semi Sync.	Semi Sync.	Sync.	Async. safe
Adversarial Model	Mildly A.	Mildly A.	Mildly-Strongly A.	Mildly A.	Mildly-Strongly A.	—	—
Economic Model	—	ϵ-Nash	—	—	—	—	—
Byz. Safety Tolerates	1/3	1/2	1/2	1/2	1/2	1/3	—

Partial synchrony assumes the existence of an upper bound on message transmission delays or the relative speed of process execution, but this upper bound is not known a priori to any nodes in the protocol. This model assumes that the messages sent are received by their recipients within some fixed time bound. In other words, while the messages may be delayed arbitrarily within the upper bound, they are guaranteed to be delivered within the time bound. [23].

Asynchrony implies that there is no fixed upper bound on how long it takes for a message to be delivered or how much time elapses between consecutive steps of a processor. Messages sent by parties may be arbitrarily delayed and no bound is assumed on the amount of time that it takes for the messages to be delivered [23].

1.4.2. Adversarial Model. Protocols assume different types of adversaries and craft defenses accordingly. In cryptography literature, the corrupted parties can be designated as static or adaptive. *Static* adversaries have corrupted a certain number of network nodes ahead of time and exercise complete control over these nodes. They are not able to change which nodes they have corrupted or to corrupt new nodes over time. In contrast, *adaptive* adversaries have the ability to control nodes and change which nodes they control to maximize their likelihood of impeding network function; that is, they adapt their corruptions as circumstances change.

There is a spectrum along which adversaries can be adaptive. *Mildly adaptive* adversaries can only corrupt parties based on past messages, and cannot alter messages already sent. Moreover, the adversary may mildly corrupt groups, but this corruption takes longer than the activity period of the group. The adversary can corrupt the proposer and block makers too if chosen ahead of time, as it can anticipate which nodes those will be. *Strongly adaptive* adversaries can see all messages sent by honest parties in any given round, and based on the message content, can decide whether or not to corrupt a party by altering its message or sabotaging message delivery. These corruptions are instantaneous.

Moreover, there are two major kinds of failure models addressed by consensus protocols: *crash failures*, when processes stop without warning and *byzantine failures*, when processes exhibit any arbitrary type of malfunction [16]. Traditional distributed consensus protocols like Paxos [28] and RAFT [29], meant to be executed in non adversarial settings only tolerate crash faults, however, blockchain consensus protocols must be robust enough to operate in adversarial settings and as such must tolerate both crash and byzantine failures.

The adversarial model that a protocol can tolerate depends on if the committee selection can be biased, predicted and if the source of randomness used by the protocol is revealed before the start of a new round. Refer to section 4 for further discussion.

1.4.3. Economic Model. Given that validators in a PoS framework receive potential rewards given for honest validation, it is impossible to divorce a PoS framework for public blockchains from its respective incentive structure. Incentive structures are developed given some assumptions of human behavior. Following traditional economic literature, protocols can be deemed incentive compatible following a Nash equilibrium or ϵ-Nash equilibrium. In a Nash equilibrium, no actor has an incentive to deviate from their behavior [3]. An epsilon Nash equilibrium approximately satisfies the constraints of a Nash equilibrium such that an actor might have a small incentive to deviate from their behavior [26]. The majority of protocols we analyze have not defined incentives and as such do not conduct a formal economic analysis of their protocol. For these protocols we are unable to comment on the economic model.

1.4.4. Analysis of Model in each Protocol. Refer to Table 1 for a succinct presentation of the network and adversarial model of each protocol discussed in this work. Dashes in the table imply incomplete information from the protocol. Here we explain the nuances of network and adversarial model for the above protocols. Unless stated otherwise, each protocol does not specify incentives or conduct a formal economic analysis of their protocol.

Tendermint operates in partial synchrony in the proposal step and asynchrony in rounds once the block is proposed. The protocol tolerates a mildly adaptive adversary controlling up to $\frac{1}{3}$ Byzantine nodes.

Thunderella has an asynchronous optimistic fast path with synchronous underlying blockchain, thus the protocol is ultimately synchronous. The protocol tolerates a mildly adaptive adversary handling adaptive security with erasures in the random oracle model. The adversary is in charge of scheduling message delivery such that they cannot modify contents broadcast by honest parties, but can reorder and delay them. An ϵ-Nash equilibrium is specified against any coalition that controls only a minority of the total computation power, such that an adversary in this case cannot earn more than its fair share of rewards. This is based on Pass and Shi's concept of fairness outlined in Fruitchains [17].

Algorand provides safety under partial synchrony such that after an asynchronous period, the network must be strongly synchronous for a reasonably long period again. The protocol ensures liveness under synchrony. Algorand tolerates an adversary between a mildly adaptive and strongly adaptive adversary in that the adversary may temporarily fully control the network and immediately corrupt users in targeted attacks because of immediate player replaceability.

Dfinity practically operates in partial synchrony, however is only formally proven in synchrony. The protocol can tolerate a mildly adaptive adversary.

Ouroboros Genesis guarantees safety under semi synchrony and tolerates an adversary between a mildly adaptive and strongly adaptive adversary in that the adversary can corrupt any participant at any moment under the assumption that the majority of the stake is held by honest nodes and the existence of an upper bound on message delay. In this model, desynchronized parties have their stake considered adversarial until some d rounds pass, and they are full synchronized again.

Casper FFG does not provide a comprehensive discussion of synchrony, but states that all clients have local clocks that are perfectly synchronized with any discrepancy treated as part of a known communication delay, thus we consider the protocol as synchronous.

Casper TFG provides asynchronous safety in that blocks won't be reverted due to the arbitrary timing of future events, and provides liveness given some synchrony assumption. The safety proof holds with arbitrary byzantine behavior.

2. General Solution Steps and Challenges

This section presents a high-level overview of the steps by which we break down the algorithms of each protocol.

2.1. Proposer and Committee election and Randomness generation

Proposer and committee election refers to the process of selecting a validators to be proposers or committee members in a round of a BFT based protocol. The proposer/committee is usually responsible for selecting the next block that is proposed and propagated to the rest of the network. Protocols deal with proposer/committee failures differently. *Random generation* refers to the mechanism by which randomness is created and agreed upon by the validator set if the protocol employs randomness in any of its components. Randomness is most often employed in proposer and committee election.

2.2. Propagation

A crucial part of consensus is for a majority of validators to decide on the same value. In order to reach agreement on a value, that particular value has to spread from its origin to the rest of the network. *Propagation* is the act of dispersing information within a network. Most protocols use a form of gossiping to take advantage of the connectivity of nodes to spread information.

2.3. Finality

Finality is the affirmation that well-formed blocks will not be revoked once committed to the blockchain. PBFT-based protocols provide absolute finality in which a transactions are immediately considered finalized once included in a block and added to the blockchain, while chain-based protocols provide probabilistic finality in which the probability that a transaction will not be reverted increases as the block which contains that transaction sinks deeper into the chain. In protocols that offer probabilistic finality, a dishonest minority can revert blocks.

2.4. Handling Churn

Churn describes change in a set of participating nodes due to joins, leaves, and failures. [21] Given the public nature of blockchains, protocols must be able to effectively deal with proposers, validators, and users joining and leaving the network.

3. Algorithms of Relevant Protocols

Here, we provide a high-level overview of the relevant protocols assuming a working knowledge of these protocols.

3.1. Tendermint

Tendermint [11] is based on an algorithm outlined by Dwork et al in their landmark paper Consensus in the Presence of Partial Synchrony [14]. A BFT-style protocol, Tendermint assumes a static committee and a proposer that is deterministically selected every round to compile and propose a block, with the rest of the committee trying to achieve a $\frac{2}{3}$ supermajority of votes for that block in two distinct rounds.

3.1.1. Proposer & Committee election and Randomness generation. Tendermint has a new proposer every round, however, this proposer is not selected through randomness. The proposer is elected via deterministic round robin between validators such that there is one proposer per round. The frequency with which a validator is selected as a proposer is equal to the validator's share of the total stake. Thus, if a validator owns $\frac{1}{4}$ of the total stake, the validator will be selected as a proposer $\frac{1}{4}$th of the time before the end of the round robin. Given the deterministic nature of proposer selection, network participants can determine who the proposer is in any given round. The committee is likewise deterministically selected and composed of validators.

3.1.2. Propagation and Creation of Blocks. Consensus proceeds in multiple rounds, each with one proposer:

1) A proposer is chosen from the validator-set to propose a new block.
2) That proposer compiles and proposes a block.
3) If $\frac{2}{3}$ validators agree on the aforementioned block (this is called the pre-vote), then the protocol moves onto the pre-commit stage.
4) If $\frac{2}{3}$ of validators agree on the same block that was pre-voted, then all honest nodes commit that block and move to new block height (starting the process at step 1 again).

If a block is not proposed in time or block does not receive enough votes in the pre-vote or pre-commit, then a new round occurs with a new block proposed at the same height.

3.1.3. Finality. A block achieves absolute finality if it receives $\frac{2}{3}$ or more pre-votes and pre-commits. This process continues indefinitely unless $\frac{1}{3}$ or more validators become unresponsive, in which case the network halts. Thus, the protocol prefers consistency over availability.

3.1.4. Handling churn. Tendermint handles rotating validator-sets by allowing updates to the validator-set by specifying public keys and updated voting power for a new set of validators. To remove validators, the protocol simply sets their voting power to 0. Validators that haven't signed for a protocol specified number of blocks, are considered timed out and implicitly unbonded. When validators stake, they lock up their funds for a particular bonding period. To unlock their funds, they must specify their intent to do so and then wait for a 2-3 month unbonding period. This allows for the knowledge of how the validator-set will change and mitigates the nothing at stake problem.

3.2. Thunderella

Thunderella [36] builds on top of Pass and Shi's Sleepy Consensus [32] and Snow White [31], to provide the basis for a scalable PoS protocol involving layering a 'fast path' which allows for optimistic instant confirmations of transactions over a slow underlying blockchain.

3.2.1. Proposer and Committee election and Randomness generation. Thunderella elects a proposer in the BFT overlay subset of the protocol, however the specific method of selecting a proposer is decided by the implementing application. Committee selection methods are also left up to the decision of the application as well, however three different options are suggested:

1) Implement a Random Oracle which is secure against slow corruptions either using a hash function and seeding it like in Snow White [31] or pseudorandom function (PRF) like in Sleepy. [32].
2) Implement a Random Oracle and VRF which is secure against adaptive corruptions using Algorand's model.
3) Use recent validators to form committee as long as the underlying blockchain is fair. This method is secure against slow corruptions.

If there are too many validators eligible to vote, the protocol considers random down-selection to reduce bandwidth consumption. This down-selection can be conducted with a random oracle or random oracle and VRF [24].

3.2.2. Propagation and Creation of Blocks. The 'fast path' of the protocol has an accelerator, which acts as the proposer in proposing blocks, and a committee, which verifies proposed blocks. If the Accelerator and $\frac{3}{4}$ of the committee is honest, then the protocol proceeds as follows. First, the Accelerator proposes transactions with associated sequence numbers. Then, the Accelerator signs transactions and sends these to the rest of the committee. If a $\frac{2}{3}$ supermajority of the committee acknowledges then the transaction is considered notarized. If either of the above does not hold, the protocol enters slow mode, falling back to the underlying blockchain and their protocol for creating blocks and propagating them.

3.2.3. Finality. Any maximum sequence of transactions with no gaps that has been notarized is considered fully confirmed output. Transactions are considered probablisitically finalized after they are some k blocks deep into the slow chain, where k is the security parameter specified by the implementing protocol.

3.2.4. Handling churn. Thunderella handles churn as it supports robust committee reconfiguration defending against posterior corruptions. Robust committee reconfiguration means that committees in Thunderella are chosen such that each committee remains honest although not necessarily online until the honest chains are roughly the clock time for the current tx(c) + 4 * security parameter(k) (c + 4k), since notarization transactions are only considered legitimate if included in the blockchain by length (c + 2k). Posterior corruptions refer to a set of users possibly holding majority of stake sometime in past selling their stake at some point and from that point onward such that they might be incentivized to act maliciously (eg. by forking and double spending old money). Thunderella suggests the following reconfiguration approaches: 1) use underlying blockchain to establish IDs of recent miners and have only recent miners form the committee or 2) have stakeholders act as committee.

3.3. Algorand

Algorand [38] is a protocol devised by Silvio Micali employing Micali's Verifiable Randomness Functions (VRFs) to choose proposers and committee members secretly and utilizes a Byzantine Agreement algorithm to reach consensus. This analysis is based on the 2017 ACM publication.

3.3.1. Proposer and Committee election and Randomness generation. Algorand chooses committee members and proposers randomly among all users based on the users weights using a technique called cryptographic sortition. This is a private non-interactive method where every validator in the system can independently determine if they are in the committee by running the VRF on their private key and info from blockchain. A *VRF* is a psuedo-random function where each output is unpredictable given the knowledge of all prior outputs. Each output has publicly verifiable proofs of output correctness. Here, the VRF generates a proof which the validator can include in their message to prove to other validators that she is in the committee. Multiple people can be selected as proposers and a person may be given multiple votes in a committee (based on their proportion of stake). In Algorand, each user is treated as a collection of unit value sub-users. The rank of a proposer is thus chosen based on the highest priority of each sub-user.

Using the VRF: $VRF_{sk(x)}$ returns two values: a hash h and a proof p. The hash h is uniquely determined by sk and x, but is indistinguishable from random to anyone that does not know sk. The proof p allows someone to check that the hash of x by person with public key pk matches the hash h without knowing sk. VRF provides these properties even if pk and sk are chosen by an attacker. Thus an attacker doesn't know if a person was chosen until they send a message which includes the hash and proof.

Selection of Committee size: The committee size is based on a calculation which depends on the number of honest validators in Algorand, the threshold of honest validators in the committee and the acceptable probability in which there will be more than the threshold number of malicious validators in the committee.

3.3.2. Propagation and Creation of Blocks. Creation and propagation proceeds as follows. There is at least one block proposed. To minimize cost of gossiping unnecessary blocks, a sortition hash is used to prioritize block proposals. (priority of a block = priority of validator who proposed block i.e the value of sub validator index that has the highest priority). Since each validator can be chosen multiple times in a round to be a proposer or in the committee, each time, the validator is given a new index and key pair. This new index and key pair is the referred to as the $sub\,validator\,index$. The agreement protocol consists of two phases:

1) BA*, a Byzantine Agreement protocol, reduces the problem of agreeing on a block to agreement on one of two options by narrowing down one non-empty proposed block to agree on.
2) 2) BA* reaches agreement on one of these options: either agreeing on a proposed block, or agreeing on an empty block.

3.3.3. Finality. In strong synchrony, BA* designates consensus on a value as final if the algorithm reached agreement in the first step, and if enough validators observed this consensus being reached. A block that is a predecessor of a finalized block also becomes finalized. Final blocks guarantee that no other block could have reached consensus in the same round. This means that all final blocks are totally ordered with respect to one another, since (1) blocks form a linear chain, and (2) there can be exactly one final block at any given position in the chain. In the case of weak synchrony, BA* may have forks and thus may only be able to reach tentative consensus. To resolve these forks, Algorand periodically proposes a fork that all validators should agree on, and uses BA* to reach consensus on whether all validators should, indeed, switch to this fork. All validators passively keep track of all forks and at every time interval they run the recovery protocol (where they agree on forks instead of blocks) to sync up the system.

3.3.4. Handling churn. Algorand handles joining in the following manner. Gossip peers are replaced every round, so if a validator gets disconnected, this replacement helps a validator recover. To help new validators catch up, Algorand generates a certificate (an aggregate of votes from the last step of BA* except the final step which allows any validator to reach the same conclusion by processing the votes) for every block that was agreed upon by BA*. Certificates speeds up the validation process of old blocks for new validators. A validator can also request a collection of votes on the final step of a block (but only really needs to ask this for the latest block because if the last block was final, everything before it was also finalized). The protocol supports leaving, assuming that most honest validators (e.g. 95%) can send messages that will be received by most other honest validators (e.g. 95%) within a known time bound. This requires most validators to be online. Further validators need to be online to know if they were chosen to be in the committee/ proposer of every round.

3.4. Dfinity

Dfinity is a consensus protocol using a decentralized key generation protocol instead of a trusted third party as a source of randomness [13]. Dfinity uses weighted proof of stake, where anyone can propose blocks but proposals are ranked based on the randomness seed for the round. The protocol employs the concept of notarizations, a version of optimistic consensus referring to a threshold signature from a majority of nodes under a block created jointly by registered clients, allowing for near instant finality. These properties ensure faster growth of the blockchain by allowing for proposals to continue to be made before previous proposals are finalized, in a way almost parallelizing proposing and validating.

3.4.1. Proposer and Committee election and Randomness generation. Dfinity generates randomness via a decentralized random beacon, using a Threshold Relay DKG scheme [41]. *Random beacon* refers to the protocol for generating randomness. Dfinity makes use of BLS signatures, distributed key generation (DKG), and threshold cryptography [10]. *BLS* refers to Boneh-Lynn-Shacham signature scheme which allows users to verify if a signer is correct [10]. *Distributed key generation* is an encryption process in which multiple parties collectively generate shared public and private keys such that the public key is outputted while the private key is shared amongst the parties via a threshold secret sharing scheme [23]. *Threshold cryptography* refers to any scheme in which a message is encrypted using a public key and the corresponding secret key is shared among n parties. To decrypt the message, at least t parties are required to cooperate in the decryption algorithm [23].

The setup proceeds in three steps: 1) the DKG algorithm is run for t of n BLS signature scheme to setup the group's public key. 2) The secret key shares are distributed. 3) There is a verification vector (v) that gets committed to the blockchain. The public key (pk) for the group (G) can be derived as follows from the verification vector: $pk_{G,i} = \prod_{j=0}^{t-1} v_j^{i^j} \epsilon G_2$ where i,j,t are indices.

Next is the signing process, which proceeds as follows. When the first notarization for round r - 1 is seen, all the validators enter the next round and try to compute the randomness using equation: $\sigma_{r,i} = Sign(r||\xi_{r-1}, sk_{G,i})$ where ξ_{r-1} is the random value of round $r-1$. When any validator receives at least t valid signatures, she can recover the randomness for that round by running the threshold recovery algorithm. The random beacon output in one round chooses the committee for the next round according to a threshold relay technique, which is the process of randomly sampling validators into groups, setting the groups up for threshold operation, choosing the current committee, and relaying from one committee to the next. The sub-committee size is chosen based on this equation: $Pr[G\ honest] \geq CDF_{binom}(\lceil n/2 \rceil - 1, n, \frac{1}{\beta})$ where $1/\beta$ is the adversarial strength.

The threshold relay is used to split validators into multiple groups. One of the α groups each of size z is chosen as the committee for beacon protocols and notarizations in that round. The committee for the beacon protocol is responsible for generating randomness for round r + 1 and the notarization committee runs the protocol.

3.4.2. Propagation and Creation of Blocks.

Propagation and creation of blocks involves the following three steps:

1) Producing a random beacon output (described above): The random beacon output for round r determines a priority ranking of all registered validators and determines the committee members.
2) Producing block proposals: The proposer first selects the heaviest (based on most notarizations) valid chain C with len $C = l$ that it knows of. Then, the new proposed block B references the last block on C and is composed of the selected transactions. This block is then broadcast to request for notarizations. Blocks will be optimistically agreed upon by all notarization committee members based on the priority ranking which the decentralized beacon decides in the notarizations round. This block is then broadcast to the other validators.
3) Producing block notarizations: A block is said to be notarized when it has majority of the notary committees signatures in that round. Block notarizations are produced by first checking the validity of a block and then signing. A proposed block B is considered valid for round r if $rd(B) = r$ and there is a valid block B' such that: 1) $prev(B) = H(B') \wedge rd(B') = rd(B-1)$, 2) $nt(B)$ is a notarization of B', and 3) $dat(B)$ is valid where $dat(B)$ is data payload. After BlockTime, each member of the notary committee signs all highest priority blocks for the current round that it has received and broadcasts a signature message for this block to the entire network. This process continues until the protocol observes a notarization for the current round. The notarization signals the start of a new round.

3.4.3. Finality.

Dfinity maintains two definitions of finality, optimistic finality through notarizations and general finality. With optimistic finality only notarized blocks can be included in a chain. Validators only notarize the highest-ranked blocks with respect to a publicly verifiable ranking algorithm driven by the random beacon. Notarization is optimistic consensus as usually only one block gets notarized and can be detected after one subsequent block plus a relay time. Whenever the broadcast network functions normally, a transaction is final in the consensus after two notarized confirmations plus a network traversal time. However, notarization is not consensus because it is possible, due to adverse timing, for more than one block to get notarized at a given height. The finalization protocol is passive and built on top the notarization protocol.

3.4.4. Handling churn.

Dfinity handles churn allowing validators to join the network (i.e. register) or leave (i.e. deregister) by submitting a special transaction. A registration transaction contains the public key of the new validator plus an endorsement proving that she was allowed to register. The registration gets activated in e + 2 epochs once the transaction is included in the blockchain. The block produced in the first round of each epoch is a registry block (also called key frame) and contains a summary of all new registrations and de-registrations of validators that happened during the previous epoch that just ended. A system parameter, M_{max}, governs how many different pools can form during an epoch. Members of the pool create a registration transaction for G which contains the tuple x = (e, j, pk_G) and has to be included in the blockchain in epoch to get registered. Pools are automatically de-registered after a certain time period.

3.5. Ouroboros Genesis

Ouroboros Genesis [18] is a Proof-of-Stake (PoS) based consensus protocol that employs a novel chain selection rule to allow robust bootstrapping of the blockchain without external knowledge. This allows new validators to verify the true longest chain with only knowledge of the genesis block. Ouroboros Genesis uses a locally evaluated lottery via VRF to select a proposer.

3.5.1. Proposer and Committee election and Randomness generation. The first step in Ouroboros Genesis is the *initialization* functionality, which is called upon by a validator to become operational. There are two cases for this depending on the block during which the functionality is called: 1) A validator that calls *initialization* during the genesis block does so to claim her stake. 2) A validator that calls *initialization* during a non-genesis block queries the functionality to receive the genesis block and uses the stake distribution to determine the threshold to potentially become a proposer.

Next, is the staking procedure in which validators locally evaluate a VRF to determine if they are elected to be the proposer. The VRF uses the validators secret key and takes as input the round index and epoch randomness. It then outputs a hash and a proof - if the hash is below a certain threshold then the validator is elected proposer (there can be multiple proposers as well). The stake distribution and epoch randomness are updated at the beginning of each epoch. Honest validators are also mandated to update their private key in each round - they do so via the key evolving signature scheme (KES) employed. The signer can invalidate the extorted signature by updating the public key. [35]

3.5.2. Propagation and Creation of Blocks. The proposer(s) creates and signs a new block and appends it to the end of the blockchain. She then broadcasts that new chain to her peers to propagate the new block. She includes the VRF output and the proof as her eligibility to be proposer. The other parties in the network can then verify that she is a proposer for this round and choose to accept the new longest chain or not.

3.5.3. Finality. In Ouroboros Genesis finality is probabilistic and dependent upon the following chain selection rule. For short range (up to k blocks): simply follow the longest chain. For long range (more than k blocks): use the plenitude rule, meaning look at the period of time right after the fork occurs off the current chain and choose the denser of the chains as the right one.

3.5.4. Handling churn. Ouroboros Genesis is built upon the idea of supporting dynamic availability. It handles churn without trusting any other validators or making any assumptions on how many validators are involved. New validators only need the genesis block to re-build the blockchain and can do so without external verification from other validators (similar to many PoW protocols). But they need to interact with other validators to synchronize with the system and continue to participate. Any validator that wants to participate in the protocol must go through a registration process, registering with the random oracle, clock, ledger. To register with the ledger, the validator must check that she is first registered with the other clock and ledger, then if so, register with other functionality in the system, and store the current time (to determine the last time this validator was connected to all resources). Deregistration is a very similar process.

3.6. Casper FFG

Casper Friendly Finality Gadget (FFG) [39] is a Proof of Stake protocol that Ethereum is in the process of employing as a transition method for switching from purely PoW system to purely PoS system. Casper FFG overlays already existing PoW systems by establishing a PoW/PoS hybrid model. It is a chain-based system that uses PoW style consensus on every block except for every 100th, which uses PoS style consensus. The PoS-style consensus uses a voting based method weighed by the staked deposits of validators in a validator-set using forward and reverse stitching of validator-sets to guarantee finality with $\frac{2}{3}$ of validator votes.

3.6.1. Proposer and Committee election and Randomness generation. FFG maintains a validator-set by locking up a validators deposit as a means of showing they have stake in the system. Anyone can self-select into the validator pool as long as they meet a minimum staking requirement. The protocol does not have a proposer and does not employ any method of randomness generation.

3.6.2. Propagation and Creation of Blocks. Casper is an overlay that helps with finality on an already existing blockchain. The actual propagation and creation of blocks is assumed to occur in the underlying system. Currently, FFG relies on the underlying PoW system, but plans to eventually move to something that is more efficient.

3.6.3. Finality. Even if a malicious attacker has control of the underlying proposal mechanism, FFG will not finalize conflicting checkpoints. FFG prevents finalization on any block or checkpoint without a $\frac{2}{3}$ supermajority by using a stitching mechanism via a forward validator-set and a rear validator-set to show how the overlap of the two ensures at least $\frac{2}{3}$ of the members of any dynamically changing validator-set agree on the same block. A *supermajority link* is an ordered pair of checkpoints (a, b) such that $\frac{2}{3}$ of validators have published votes with that pair. The source(a) and target(b) do not need to be contiguous.

3.6.4. Handling churn. Dynamic validator-sets are handled using the idea of a *dynasty* of a block, the number of finalized checkpoints from the root to the parent of the current block. A new validator is required to include a deposit for their stake in a block with dynasty d. That new validator will join the validator-set at the first block with dynasty $d + 2$. Similarly, a retiring validator must include a withdraw in block with dynasty d and will be removed from the validator-set at dynasty $d + 2$. Any validator that leaves the validator-set will have their corresponding public key forbidden from rejoining that validator-set. This is done as a means of reducing overhead in terms of handling churn, not as a form of punishment. At the start of block with dynasty $d + 2$, the retiring validator's stake is locked up on the order of 3-4 months worth of blocks as a safety guarantee. Any violation of protocol policy will cause these funds to be slashed.

3.7. Casper TFG

Casper CBC is a family of "Correct by Construction" consensus protocols developed by Vlad Zamfir at Ethereum. Casper The Friendly Ghost (TFG) [40] is the protocol amidst this set of protocols which applies the CBC framework to blockchain consensus. TFG is currently a work in progress, with a comprehensive safety framework and finality provisions in that nodes make decisions on safe estimates, but in depth details regarding liveness and protocol specifics are being developed for later inclusion.

3.7.1. Proposer and Committee election and Randomness generation. TFG does not specify any particular method of proposer or committee election method, and does not rely on random generation although some liveness strategies that they may employ may require some randomness.

3.7.2. Propagation and Creation of Blocks. Validators create blocks according to any strategy ensuring liveness with some synchrony assumption and less than some number of faulty nodes. Blocks are gossiped from there. No specific recommendations are currently provided on a strategy for block creation ensuring the above parameters although some validator strategies are in the works.

3.7.3. Finality. Any two validators can make final decisions as soon as they detect safety on blocks that would require greater than their fault tolerance threshold weight of validators to equivocate before a conflicting block could be finalized. When validators equivocate, a fault tolerance threshold is reached, so if the amount of safety on a block is greater than the validators fault tolerance threshold, the node is safe to make a decision on it. Fault tolerance thresholds are subjective in that different validators can have different fault tolerance thresholds. In the case two validators have different fault tolerance thresholds f and f', then they have consensus safety if the union of their views has less than min(f, f') faults.

3.7.4. Handling churn. TFG handles validator-set rotation with dynamically changing sets of consensus forming nodes and / or their weights. The safety proof holds with dynamic validator-sets. Future validators cannot affect the fork choice of blocks before they were introduced, or they would affect the safety of those blocks. Thus, the validator selection method adds a weight map to each block that specifies the validator-set for the fork choice from among its children, as they cannot affect blocks in the past. This allows for the rotation of validator-sets without requiring that the rotation is finalized.

4. Security of randomness of different schemes

This section analyzes the randomness generation of each of the protocols. Randomness is an important part of every protocol because it impacts committee and proposer selection. A predictable or biasable source of randomness would reduce the strength of an adversary that the protocol can tolerate. Hence it is important to understand how an adversary can benefit from the randomness scheme that a protocol uses. As such, we look at two kinds of adversaries: an active adversary and a passive adversary [23]. A *passive* adversary is one who would try to predict the outcome of the randomness, without interacting with the randomness scheme in any manner i.e the adversary is merely capable of observing all the interactions of other users with the randomness scheme. A protocol which has *predictable* randomness means that given the inputs to the randomness scheme, there exists some stage in the protocol in which the output of the randomness generation function does not appear completely random to an adversary i.e some output values are more likely than other output values. More formally, no party learns anything about rnd_r, except with negligible probability, until they have gathered at least t randomness shares in round r. While having an unpredictable source of randomness plays a significant role in securing a protocol, a good randomness scheme should also be able to defend against an active adversary. An *active* adversary tries to tamper with the protocol by modifying the inputs to the randomness generation function with the goal of changing the resulting randomness output for a round. This is referred to as *biasable*. More formally, the output rnd_r of round r represents an unbiased, uniformly random value, except with negligible probability. The adversary may or may not be able to determine the resulting randomness seed. Thus the ability of an adversary to bias the randomness seed is independent of the adversary's ability to predict the randomness function's output. It is also useful to understand if a protocol reveals the randomness seed before the current round because leaking of the seed ahead of time enables the adversary to learn the proposers and committee ahead of time. This information can be used by an adversary in an attempt to corrupt the chosen committee or proposers from the time when the seed was leaked to the time the new round starts. Table 2 provides a succinct comparison of these properties. Dashes in the table imply incomplete information from the protocol.

It is also important to note that Random Oracle is a theoretical construction. In practical implementations, only a Pseudorandom Function is available and is used as an approximation of the Random Oracle.

4.1. Biasable and Predictable

In this section we explain why the randomness schemes of each of the protocols can be biased or are bias resistant. We describe the randomness schemes for Tendermint, Algorand, Dfinity, and Casper FFG. Thunderella and Ouroboros use a randomness scheme similar to what Algorand proposes. Thunderella also suggests another randomness scheme described in Snow White which we analyze below. Casper TFG doesn't have a proposed randomness scheme yet.

TABLE 2. SECURITY OF RANDOMNESS MODEL

	Tendermint	Thunderella	Algorand	Dfinity	Ouoroboros Genesis	Caspser FFG	Casper TFG
Predictable	Yes	Yes	Yes	No	Yes	Yes	—
Biasable	Yes	Yes	Yes	No	Yes	Yes	—
Revealed before current round	Yes	Y/N	No	Yes	No	Yes	—

4.1.1. Tendermint.

Tendermint uses a deterministic round-robin protocol scheme to choose proposers; there is no randomness in the protocol. Proposers are elected deterministically through a heap sort based on the voting power and number of times the validator was chosen. The protocol can only be biased by adding or removing stake since that is the only input the adversary can access. The protocol cannot be biased instantaneously because of the long time it takes for a validator to *unbond* i.e remove stake from the system or *bond* i.e add stake to the system. Nevertheless, an attacker can plan ahead to bias the choice of proposer in a longer time frame. Since Tendermint uses a deterministic round robin algorithm, one can predict who the proposer could be for that round.

4.1.2. Algorand.

The randomness scheme in Algorand is as below: each validator v selected as a proposer computes a seed for round r using the equation $< seed_r, \pi > \leftarrow$ $\mathrm{VRF}_{sk_v}(seed_{r-1}||r)$ where sk_v is the secret key of validator v and $seed_{r-1}$ is the randomness seed of the previous round.

The VRF proof π and the seed for round r are contained in the block accepted in round $r-1$. If the given VRF proof doesn't verify the given seed, then everyone computes the seed for the new round as $\mathrm{H}(seed_{r-1}||r)$ where H is a hash function. This means that the secret key of each validator has to be chosen well in advance to ensure that they can't bias the seed.

When the network is not strongly synchronous, the attacker has complete control over message passing links, and can therefore drop block proposals and force users to agree on empty blocks such that future selection seeds can be computed. Algorand thus requires the *weak synchrony* assumption that in every period of length u, there must be a strongly synchronous period s of length $s < u$ for the protocol to be bias resistant. As long as s is suitably large enough that at least one honest block was produced in a time period of b, an adversary v' choosing a key $sk_{v'}$ cannot bias the randomness seed for round r.

In all scenarios that the randomness is biasable, it is also slightly predictable since it enables the adversary to reduce the probability of occurence of certain outputs. The probability that a permissioned network is weakly synchronous is very low, thus the above scheme is for the most part unbiasable.

4.1.3. Dfinity.

In several protocols adversaries usually abort the protocol to invoke a fallback mechanism, and thus introduce bias. The threshold scheme that Dfinity uses is bias resistant because the threshold is chosen in a way that ensures an adversary cannot abort the protocol by preventing a *threshold signature* from which the randomness seed is derived from being created. This requires t the threshold to be chosen according to the equation: $t \in [f+1, n-f]$ where f is the number of signatures the adversary controls, n is the total number of signatures in the scheme, and t is the threshold of signatures required to generate randomness. This threshold is chosen to ensure that an adversary cannot predict the outcome of creation of a signature nor can the adversary prevent its creation.

One caveat is that if an adversary has more than 50% of all deposits in any BLS scheme they would be able to manipulate the final signature and randomness. However if an adversary had such a huge portion of stake there are other adversarial attacks and this breaks the fundamental assumptions of the Dfinity protocol.

4.1.4. Thunderella.

In the Random Oracle scheme instantiated with a hash function described in [31], a proposer is determined by the equation: $\mathrm{H}^{nonce}(pk, q) < D_p$ where H is the hash function which is used as the random oracle, pk is the public key of the validator, q is the given time-step and $nonce$ is the source of entropy to the hash function. The $nonce$ is chosen by the proposer of the previous block. The difficulty parameter D_p is set such that a committee member is elected as the proposer with probability w in a single time step.

If the adversary proposes a block, she can bias the nonce used to generate entropy for the hash function of the next round, thus biasing the proposer elected in the next block. However, to mitigate the biasability of the randomness scheme, the same nonce is used in the hash function to select proposers for r rounds rather than to merely select the next proposer. This makes it computationally harder for an adversary to brute force a nonce that will enable them to be the proposer for the next r rounds. While this strategy guarantees only a polynomial security loss, it comes with a predictability trade-off. This scheme results in being able to tolerate only a slow adaptive adversary

In the above algorithm, when the same nonce is reused to seed the hash function it leads to an adversary being able to predict the proposers ahead of time.

4.1.5. Casper FFG.

The randao scheme that Casper FFG plans to use is dependent this algorithm: Every validator at the start of an epoch commits $H(H(H(..S_v))))$ where S is the seed the validator commits to. $R := R \oplus$ pre-image of the inner layer of hash. At a round, a validator can either make a block or abort.

If in the fallback mechanism no randomness is generated then that appears to be a bigger problem than the randomness being predictable or biasable because then you no longer need a $\frac{1}{3}$rd malicious to abort liveness, one person is sufficient.

4.2. Revealed before the current round

In some protocols the randomness seed is revealed before the new round starts. The revelation of the seed can give any adversary an unfair advantage of being able to identify the block proposers for the round ahead of time and DoS them. This is an important factor to analyze since it determines the strength of the adversary the protocol can defend against. The analysis of Tendermint, Algorand, Dfinity, Casper FFG are as below. Ouroboros and Thunderella use the same source of randomness as Algorand while Casper CBC hasn't decided its source of randomness. An analysis of Thunderella when the protocol uses a Random Oracle instantiated as a hash function is also included below.

4.2.1. Tendermint. Use of a deterministic random algorithm means the randomness seed is revealed well ahead of every round and the proposer can be determined ahead of time.

4.2.2. Algorand. Only once a block has been proposed is the new seed and a VRF proof to verify it publicly known. This ensures that the proposer and the randomness seed is not leaked ahead of time. This guarantee ensures that Algorand can defend against DoS attacks on proposers and is adaptively secure in a model with erasures, even with instantaneous corruption.

4.2.3. Dfinity. The randomness seed is revealed once any honest validator has advanced to round r. While the time gap between an honest validator advancing and the new round officially starting is small, this gap is sufficient for an adversary with significant computational resources to identify and DoS proposers. This is the reason Dfinity can only tolerate mildly adaptive adversaries and not instantaneous corruptions.

4.2.4. Thunderella. Since the same nonce is reused as entropy during an epoch, it leads to the randomness seed being leaked ahead of the start of a new round. This enables an adversary to corrupt or DoS proposers.

4.2.5. Casper FFG. The seed for the next round is known to the validator who is the current proposer.

5. Results and Analysis

Here we present the results provided by these different protocols and an analysis of each. The aforementioned protocols will differ in their performance and ability to scale to more users when considering theory vs. practical implementation. We focus on finalization guarantees, scalability, a comparison of handling churn, and behavior in the case of a network partition.

5.1. Finality Guarantees

Here we examine the finality guarantees each protocol provides. Different types of systems provide fundamentally different finalization assurances.

5.1.1. Tendermint. Any block that has received both $\frac{2}{3}$ or more pre-votes and pre-commits is finalized.

5.1.2. Thunderella. Finality is different from confirmation and is fundamentally contingent on the realization of the underlying state replication protocol. Confirmation means commitment to publication in the underlying slow chain. Any maximum sequence of transactions with no gaps that has been notarized is considered fully confirmed output. If $\frac{3}{4}$ of the fast-path committee are honest and online and the proposer is honest then valid transactions are instantly confirmed. However, this fast path confirmation is optimistic finality, while transactions are only fully finalized once they are recorded on the underlying blockchain.

5.1.3. Algorand. The protocol achieves probabilistic finality. As long as the attacker controls less than $\frac{1}{3}$ of the monetary value, Algorand can guarantee that the probability for forks is negligible, as this allows the protocol to operate in strong synchrony, reaching definitive agreement on each block. In weak synchrony, Algorand may fork, but uses BA* to come to agreement on which fork to choose, periodically proposing a fork for the committee to vote on. As such, transactions in Algorand are eventually finalized when the protocol returns to strong synchrony. Moreover, Algorand is able to experimentally deduce that when there is an 80% fraction of honest users, an adversary would need 20% of Algorand currency to create a fork.

5.1.4. Dfinity. The probability of finality increases as the block weight on a chain increases. This assumes that for each round r, there is a time when we can rule out any further notarized blocks being received. At this rule out time, we can finalize round r because we know that the notarized blocks already contain all the chain tips that can possibly survive beyond round r. There is a guarantee of near-instant finality as under normal operation in round r, every transaction included in a block for round r is final for an observer after two confirmations plus the maximum network roundtrip time 2Δ.

5.1.5. Ouroboros Genesis. The protocol achieves probabilistic finality based on the chain-selection rule. The paper includes pseudocode for a simulator which is used for the provided proofs.

5.1.6. Casper FFG. Finality is achieved when a $\frac{2}{3}$ supermajority of the committee weighed by stake signs a block (given that fewer than $\frac{1}{3}$ of committee are Byzantine). Casper FFG is built such that it will never finalize conflicting checkpoints even if an adversary has control over the underlying blockchain's proposal mechanism. However, since

TABLE 3. RESULTS

	Tendermint	Thunderella	Algorand	Dfinity	Ouroboros Genesis	Caspser FFG	Casper TFG
Finality Guarantees	Abs.	Prob.	Prob.	Prob.	Prob.	Prob.	Abs.
Handles Churn	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Network Partition Resolution	consistent	available	consistent	available	consistent	available	consistent

FFG provides safety and the proposal mechanism provides liveness, the aforementioned adversary could prevent Casper from finalizing future checkpoints by stalling consensus.

5.1.7. Casper TFG. TFG achieves finality under validators with different fault tolerance thresholds. That is, the protocol is asynchronously safe and BFT, allowing for validators to have different fault tolerance thresholds.

5.2. Scalability Comparison

Here we look into scalability as the capability of a system to handle growth efficiently, measuring such via latency and throughput of blocks and transaction finalization. The issue of scalability in blockchain systems is critical and is one that many protocols have attempted to address, favoring PoS as a sybil control mechanism to scale by removing PoW's computational overhead. .

The majority of protocols we consider including Tendermint, Thunderella, Dfinity, Ouroboros Genesis, Casper FFG, and Casper TFG do not publicly specify latency or throughput experiments. Algorand, did conduct several protocol experiments and provides the results in their paper. They conducted these experiments on 1,000 EC2 VMs simulating up to 500,000 users. Thee results convey that for all users in at least one part of a connected component in a graph, message passing time grows with the diameter of that component, which is logarithmic in the number of users. They also show that latency increases as block size increases per round and that with 50k users and 1000 VMs an expensive round can be completed in around 100 seconds. The lowest throughput is 2Mb block in 22 seconds.

One way to achieve scalability in decentralized systems is to run the protocols mostly inside a randomly selected committee instead of the whole network [42]. This pattern began with [43] in 2007, and Tendermint, Thunderella, Dfinity, Casper FFG, and Casper TFG follow suit.

5.3. Handling Churn Comparison

Here we compare ways in which different protocols handle churn with regard to validators joining and leaving the network. Each of these protocols handle churn to some degree, with some allowing for more churn than others.

5.3.1. Tendermint. Tendermint allows for rotating committees, however, the requirement for locking periods (validators bound to the system for a 2-3 month period), means the protocol allows for limited churn. Moreover, offline validators are considered timed out and are implicitly unbonded, removing them from the validator-set, unlike sleepy protocols, in which offline nodes are still considered to be part of the validator-set, allowing for greater flexibility. [32]

5.3.2. Thunderella. Thunderella allows for committee rotation supporting sleepy consensus in that committees are chosen such that each committee remains honest although not necessarily online until the honest chains are roughly the clock time for the current txn(c) + 4 * security parameter(k) (c + 4k). As such, Thunderella supports more flexibility in churn than protocols that don't support the sleepy paradigm.

5.3.3. Algorand. Algorand allows for high flexibility in terms of joining the protocol. However, to achieve liveness in strong synchrony, it requires that at least 95% of validators must be online. This means that leaving the protocol has a relatively low churn rate.

5.3.4. Dfinity. Dfinity allows validators to request to join and leave within e + 2 epochs once the special deregistration transaction has been submitted. As such, Dfinity supports flexible churn but validators must wait a certain period of time before being able to join or leave. Moreover, offline validators are considered adversarial as validators must submit special transactions in order to leave.

5.3.5. Ouroboros Genesis. In Genesis, validators only need the genesis block to re-build blockchain without external verification (similar to many PoW protocols), but they need to interact with external parties to synchronise with the system and continue to participate. To join, validators must complete a registration process and to leave, they must go through a similar deregistration process. As such, Genesis allows for flexibility in churn depending on how long the registration and deregistration processes take.

5.3.6. Casper FFG. Casper FFG allows for validators to join and leave the protocol within a certain time period. Validators that deposit in dynasty d are added in dynasty d + 2 and validators that exit in dynasty d are removed in dynasty d + 2. When validators choose to leave, their funds are locked up for about 3 - 4 months. FFG is similar to Tendermint in that it allows for dynamic churn within the constraints of lock up periods.

5.3.7. Casper TFG. TFG handles validator-set rotation with dynamically changing sets of consensus forming nodes and/or their weights in the case that the validator selection method adds a weight map to each block that specifies the validator-set for the fork choice from among its children. This allows for the rotation of validator-sets without requiring that the rotation is finalized. As such, TFG handles churn in that a different validator-set is allowed per fork choice.

5.4. Network Partition

Here we consider how the protocol reacts to a network partition, prioritizing either consistency or availability.

5.4.1. Tendermint. Tendermint prioritizes consistency over availability. That is, no additional blocks will be confirmed or finalized until the partition is resolved.

5.4.2. Thunderella. The fast path conditions fail to be upheld and the protocol would fall back to the underlying blockchain, upholding the underlying blockchain's partition resolution conditions. As Thunderella falls back to a synchronous underlying blockchain, it prioritizes availability as synchronous blockchains are not consistent in the case of a network partition.

5.4.3. Algorand. Algorand ensures safety, implying that it prioritizes consistency over availability. Indeed, Algorand prefers to produce empty blocks generating unproductive liveness, rather than sacrificing on safety.

5.4.4. Dfinity. Network partitions are implicitly detectable by the protocol, since if the network splits in two halves of about the same size, this will automatically cause the random beacon to pause within a few blocks so that none of the sides can continue, prioritizing consistency. The random beacon will automatically resume once the network reconnects. If the network splits in a way that one component is larger than half of the network, the protocol may continue in that one large component but will pause in all other components. This assumes that a group is a good sampling of the network.

5.4.5. Ouroboros Genesis. A node that is out of sync with the protocol will be considered adversarial until at least some function of Δ rounds has passed. Thus, it seems as though the protocol prioritizes consistency.

5.4.6. Casper FFG. FFG drains non-active users of their stake over time to recover liveness. FFG prioritizes consistency as it does not allow for checkpoint finalization without a $\frac{2}{3}$ supermajority of validators.

5.4.7. Casper TFG. In the event of a network partition, Casper TFG prioritizes availability, however, validators are unable to detect safety on blocks made during a partition thus there is the chance of reversion. Safety in TFG is dependent on the maximum length it takes for the safety oracle to determine the estimate as safe. In asynchrony, this is unbounded, but with any liveness strategy that works in at least a partially synchronous network, it is bounded.

6. Conclusion

Given the breadth of knowledge required for creating and deploying alternative consensus protocols, it is imperative to have a structured framework to objectively examine the characteristics of each. We have provided a clear framework in which to compare these consensus protocols. Generalizing these protocols in a way to systematically compare their properties, we distinguish their most critical features in proposer and committee election including the corresponding randomness protocols, block propagation, and block finalization. We consider the environments, and models in which these protocols operate, looking at their network, adversarial, and economic assumptions. We also provide an evaluation of the results of these protocols in their finalization guarantees, scalability capabilities, ability to handle churn, and network partition resolution. This scheme can be used to evaluate the parameters of a single system, and in turn, compare those parameters across various systems.

7. Future Work

There remains substantial research to be done in the alternative consensus space. Many of these alternative consensus protocols provide particular guarantees, but still stand to grow stronger. In Thunderella, with greater than $\frac{1}{3}$ malicious actors, the protocol essentially always falls to the slow chain. In Algorand, if the protocol falls into elongated periods of asynchrony, it simply produces null blocks. In Dfinity, asynchrony could ensure that no block is ever truly finalized. In Tendermint, the deterministic round robin scheme may be vulnerable in practice because revealing the randomness source ahead of time enables DoS attacks on the validators. And both Casper FFG and Casper TFG require formal protocol specifications in multiple areas of their schemes.

Moreover, there is a lack of research into the economics and game theory of these systems, particularly in incentive structures. This is a crucial piece to optimally parametrize these protocols and create sustainable blockchains.

Once many of these protocols are deployed in practice, it would be valuable to revisit these analyses and understand the implications of practical implementation versus the theoretical formation of these protocols.

Further, as the space develops, it is fundamentally important to maintain high standards of technical rigor in creating these complex systems, which is made possible by having a common, comprehensive framework within which to evaluate these systems. We look forward to seeing continued work in these kinds of analyses.

Acknowledgments

The authors would like to thank Alberto Sonnino of UCL, Ethan Buchman and Sunny Aggarwal of Tendermint, Vlad Zamfir, Nate Rush, and Vitalik Buterin of the Ethereum Foundation, Professors Elaine Shi and Rafael Pass of Cornell and CornellTech, and Aditya Sripal from FourthState.

References

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

[2] G. Wood, "Ethereum: a secure decentralized generalized transaction ledger." Ethereum project yellow paper 151 (2014): 1-32.

[3] N. Nisan, T. Roughgarden, E. Tardos, V.V. Vazirani, *Algorithmic Game Theory*, Cambridge, UK: Cambridge University Press, 2007.

[4] Y. Gilad, R. Hemo, S. Micali, "Algorand: scaling byzantine agreements for cryptocurrencies." *Proc. of the 26th Symp. on Op. Sys. Principles*. ACM, 2017.

[5] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics.* Hoboken, NJ: Wiley, 2004.

[6] H. Attiya and N. Lynch, "Time bounds for real-time process control in the presence of timing uncertainty," *[1989] Proc. Real-Time Systems Symposium, 5-7 Dec. 1989, Santa Monica, CA, USA,* IEEE: 1989.

[7] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis. "SoK: Consensus in the age of blockchains," arXiv: 2017.

[8] Frequently Asked Questions. Bitcoin - Open Source P2P Money.

[9] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "SoK: Research perspectives and challenges for bitcoin and cryptocurrencies," *IEEE Symposium on Security and Privacy, San Jose, CA, USA 17-21 May 2015*, 2015.

[10] D. Boneh, B. Lynn, H. Shacham, Short signatures from the weil pairing, Journal of Cryptology, vol. 17, no. 4, 2004.

[11] E. Buchman, "Tendermint: byzantine fault tolerance in the age of blockchains," University of Guelph: 2016.

[12] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, and E. Gun. On Scaling Decentralized Blockchains, in *International Conf. on Financial Crypto. and Data Security, 2016*, Clark J., Meiklejohn S., Ryan P., Wallach D., Brenner M., Rohloff K. (eds) Financial Cryptography and Data Security. FC 2016., vol 9604. pp 106-125. Springer: Berlin, Heidelberg.

[13] T. Hanke, M. Mohavedi, D. Williams, DFINITY Technology Overview Series: Consensus System, DFINITY Stiftung, 2017.

[14] C. Dwork, N. Lynch, and L. Stockmeyer, Consensus in the presence of partial synchrony, Journal of the Assoc. for Computing Machinery, pp. 288-323, Apr. 1988.

[15] Ethereum proof of stake FAQ, Ethereum/Wiki. GitHub.

[16] M. Fisher, N. Lynch, M. Patterson. "Impossibility of distributed consensus with one faulty process," Journal of the Assoc. for Computing Machinery, Vol. 32, No. 2, April 1985, pp. 374-382.

[17] R. Pass and E. Shi, "FruitChains: a fair blockchain" Cornell, 2017.

[18] C. Badertscher, P. Gazi, A Kiayias, A Russell, and V. Zikas, Ouroboros Genesis: composable proof-of-stake blockchains with dynamic availability, IOHK: 2018.

[19] A. Gervais, G. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains." *Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. ACM*, 2016.

[20] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News, vol. 33, no. 2, 2002, p. 51.

[21] P.B. Godfrey, S. Shenker, and I. Stoica, "Minimizing churn in distributed systems." SIGCOMM '06 Oct. 2006, p.147-158.

[22] L. Lamport, "Proving the correctness of multiprocess programs," *IEEE Tran. on Software Eng.*, pp. 125-143, 1977.

[23] M. Zamadi, M. Mohavedi, "Crypto Reference, unpublished.

[24] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions." *40th Annual Symp. on Foundations of Comp. Sci., New York, NY, USA, 17-19 October 1999*, 1999.

[25] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol." Annual International Cryptology Conference. Springer, Cham, 2017.

[26] C. Papadimitriou, CS 294 Topics in algorithmic game theory: approximate nash equilibria. Berkeley EECS, Sept. 2011.

[27] V. Buterin, Parameterizing Casper, 2017.

[28] L. Lamport. "Paxos made simple." ACM Sigact News 32.4 (2001): 18-25.

[29] Agrawal, Prathima. "RAFT: A recursive algorithm for fault tolerance." ICPP. 1985.

[30] R. Pass, and E. Shi. "Rethinking large-scale consensus, in *2017 IEEE 30th Computer Security Foundations Symposium (CSF), Santa Barbara, CA, USA, 21-25 August, 2017*, IEEE: 2017.

[31] P. Daian, R. Pass, and E. Shi, "Snow White: robustly reconfigurable consensus and applications to provably secure proofs of stake." Cornell, 2016.

[32] R. Pass and E. Shi, "The sleepy model of consensus." Cornell, 2017.

[33] M. Vukolic. "The Quest for scalable blockchain fabric: proof-of-work vs. BFT replication." *International Workshop on Open Problems in Network Security*, pages 112-125. Springer, 2015.

[34] A. Zohar. Securing and Scaling Cryptocurrencies. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, IJCAI-17, pages 5161-5165, 2017

[35] G. Itkis and P. Xie. Generalized Key-Evolving Signature Schemes or How to Foil an Armed Adversary, Applied Cryptography and Network Security 2003, Lecture Notes in Computer Science 2846, pp. 151-168, 2003.

[36] R. Pass and E. Shi, "Thunderella," Cornell: 2017.

[37] J. Kwon, "Tendermint: Consensus without mining." Draft v. 0.6, fall (2014).

[38] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, N. Zeldovich, "Algorand: Scaling Byzantine Agreements for Cryptocurrencies," MIT CSAIL: 2017.

[39] V. Buterin and V. Griffith, "Casper the Friendly Finality Gadget", arXiv: 2017.

[40] V. Zamfir, Casper the Friendly Ghost: A 'Correct by Construction' Blockchain Consensus Protocol," Ethereum Foundation Github: 2017.

[41] P. Feldman, A practical scheme for non-interactive verifiable secret sharing. IEEE Symposium on Foundations of Computer Science, pages 427–437. IEEE, 1987. doi:10.1109/SFCS.1987.4

[42] B. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani, "Fast asynchronous byzantine agreement and leader election with full information" *Symposium on Discrete Algorithms (SODA)*, 2008.

[43] V. King, J. Saia, V. Sanwalani, and E. Vee, "Scalable leader election", in *Symposium on Discrete Algorithms*, 2006.