Survey of Consensus Protocols and Scalability Solutions

Pascal Berrang[1]

***Abstract*—The area of distributed ledgers is a vast and quickly developing landscape of consensus protocols. This makes it considerably hard to gain a comprehensive overview of the current state of protocols. Many of those trade one property for another and there does not seem to be a widely accepted and clear winner in the race for the best protocol.**

This paper aims at providing an overview of most of the recent directions in this landscape, not claiming completeness. Instead of completeness, it targets to deliver a solid basis for discussions about current and future developments in the field of distributed ledgers, emphasising scalability. Besides purely consensus-based scalability and sharding, we also explore off-chain scalability solutions and recent advances and implications in privacy-preserving protocols.

I. Introduction

Blockchains or distributed ledgers in general and cryptocurrencies, in particular, are instances of replicated state machines. In a cryptocurrency, clients generate and submit transactions. A network of nodes receives and processes them, thereby changing the replicated state according to predefined rules. When abstracting away the implementation details, the problem ultimately boils down to building a globally-consistent, totally-ordered, append-only transaction log. In traditional literature, such a primitive is called *total order* or *atomic broadcast* [1].

In such a system, nodes receive transactions as input and have the goal to agree on the ordering of these transactions. Informally, we require two main properties from such a primitive: *liveness* and *safety*, which can be divided into further properties as discussed in other works [2], [3]. Liveness describes the property that requests from a correct client will be eventually processed, whereas safety (or *consistency*) describes that if one honest client accepts a value, then all other honest nodes make the same decision.

The area of distributed ledgers is a vast and quickly developing landscape of consensus protocols. This makes it considerably hard to gain a comprehensive overview of the current state of protocols. Many of those trade one property for another and there does not seem to be a widely accepted and clear winner in the race for the best protocol.

This paper aims at providing an overview over most of the recent directions in this landscape, not claiming completeness. Instead of completeness, it targets to deliver a solid basis for discussions about current and future developments in the field of distributed ledgers.

We emphasise scalability, which considers transaction throughput as one of the most important properties. That is because, generally, a property that distinguishes payment networks from traditional consensus systems is that the former are not so much latency critical, but favour a high throughput. For example, the Visa network processes 2,000 transactions per second (tx/sec) on average, with a peak capacity of 56,000 tx/sec[1].

In this paper, we divide today's advances in consensus related research into four categories:

1) traditional consensus protocols (mostly in closed settings, e.g. PBFT [4]),
2) Proof-of-Work based consensus protocols (e.g. Bitcoin),
3) Proof-of-X based consensus protocols (replacing PoW with more energy efficient alternatives),
4) and hybrid protocols that constitute compositions of the aforementioned categories.

From these categories, we also derive our organisation as described below.

Besides purely consensus-based scalability and sharding, we also explore off-chain scalability solutions and recent advances and implications in privacy-preserving protocols.

A. Organisation

We start by introducing some of the basic results in consensus theory in Section II, before we explain the basic building blocks in Section III – spanning traditional consensus protocols, Proof-of-Work, and Proof-of-X. In Section IV, we discuss how protocols for a closed set of participants can be transformed into open networks, providing the basis for hybrid protocols.

Then, we organise the protocols presented in this paper into further categories in Section V, which are Proof-of-Work-based protocols in Section VI, Proof-of-Stake-based protocols in Section VII, hybrid protocols in Section VIII, DAG-based protocols in Section IX, protocols leveraging a chain per account in Section X, leaderless protocols in Section XI, and finally sharding protocols in Section XII.

We also explore the direction of off-chain scalability solutions in Section XIV and discuss advances and implications of privacy-preserving protocols in Section XV.

Lastly, we refer the reader to interesting, related literature in Section XVI and conclude in Section XVII.

II. Basic Results

There are two basic results in consensus theory, which motivate most of the trade-offs. One states the impossibility of achieving consensus in a fully asynchronous system, the other one gives an upper limit on the number of Byzantine nodes in a deterministic, reliable system.

Before discussing these results, we first need to understand what it means for a system to be synchronous or asynchronous.

[1]Nimiq, pascal@nimiq.com

[1]See Visa Fact Sheet 2015: https://usa.visa.com/dam/VCOM/download/corporate/media/visa-fact-sheet-Jun2015.pdf

1) **Synchrony:** In a synchronous system, messages will always arrive within a maximum, known delay of Δ.
2) **Asynchrony:** In an asynchronous system, messages may be delayed arbitrarily, and there does not exist a reliable bound.
3) **Partial/Weak Synchrony:** A partial or weak synchronous system assumes the network to become eventually synchronous, despite potentially a long period of asynchrony.

A. FLP Impossibility Result

The FLP impossibility result proves the impossibility of achieving and maintaining consensus in a system of distributed, deterministic asynchronous processes if even a single process could crash [5]. There are two ways to overcome this impossibility result:

1) using weak or partial synchrony (two equivalent properties), often also relying on *leaders* who can be skipped if they are suspected of misbehaving or not responding,
2) or using non-determinism and randomisation.

While the second option is typically slower (but requires less communication), recent advances have been made that achieve considerable improvements in speed (e.g. HoneyBadgerBFT [3]).

Practical Byzantine Fault Tolerance [4] is one of the most prominent examples overcoming the impossibility result using leaders. However, PBFT is not suitable for a cryptocurrency scenario without further modifications. An example of the second category is Bitcoin's Nakamoto consensus, which employs a Proof-of-Work algorithm to make consensus probabilistic.

B. The Byzantine General's Problem

The Byzantine General's Problem describes the problem of creating a reliable, deterministic, decentralised consensus system in which Byzantine nodes can provide contradicting information (e.g. voting on different blocks at the same height). The corresponding paper by Lamport et al. [6] proves that such a system can only exist if at most $\lfloor \frac{n-1}{3} \rfloor$ nodes are Byzantine.

III. Basic Building Blocks

Achieving consensus is not a modern problem. While there are many recent advances and a lot of new requirements shaped by the cryptocurrency and blockchain industry, the general question has been discussed for decades. In order to understand many of today's protocols, it is necessary to explain a few of the basic building blocks and algorithms.

In particular, we discuss the building blocks of the following categories of protocols:

1) classical consensus protocols (mostly in closed settings) with the distinction between
   - Byzantine fault tolerance
   - and non-Byzantine fault tolerance,
2) Proof-of-Work-based consensus protocols (e.g. Bitcoin),
3) Proof-of-X-based consensus protocols (replacing PoW with more energy efficient alternatives) including
   - Proof-of-Stake
   - and other alternatives,
4) and hybrid protocols that constitute compositions of the aforementioned categories.

A. Practical Byzantine Fault Tolerance

As the name states, Practical Byzantine Fault Tolerance [4] (PBFT) was the first system designed to provide Byzantine Fault Tolerance in a fast and practical way. It is based on the advancements made in the DLS paper [7] (see Section XVIII-A). Previous algorithms either assumed an unrealistic, synchronous system or were too slow to work in practice. The PBFT algorithm allows replicating a deterministic state machine including its common state in a network of n nodes. It offers both liveness and safety provided at most $\lfloor \frac{n-1}{3} \rfloor$ out of n nodes are faulty. Or equivalently, if f nodes are considered Byzantine, it offers resilience against Byzantine faults for $n \geq 3f+1$ Therefore, the resiliency of this algorithm is optimal for a BFT protocol.

The system model of PBFT assumes an asynchronous distributed system, in which *the participants are a closed and well-defined set of nodes*. One of these nodes is the *leader*; the other nodes are called *backups*. The network may fail to deliver messages, delay them, duplicate them, or deliver them in a different order. In general, BFT algorithms are optimised to work in deployment scenarios in which CPU and latency are the main bottlenecks.

The algorithm works in three phases:

1) **Pre-Prepare:** The leader sends out the information that must be agreed upon to the other nodes.
2) **Prepare:** Each node checks the validity of the *pre-prepare* and, upon success, sends back a *prepare* message.
3) **Commit:** After receiving at least $2f + 1$ *prepare* messages, a node enters the *prepared* state and sends out a *commit* message.

Then, after receiving at least $2f + 1$ *commit* messages, the state change is executed. To overcome the FLP impossibility result, the liveness property requires a form of weak or partial synchrony, which is achieved through delays and timeouts.

The leader in PBFT generally remains in his role and is replaced only if suspected faulty. In this case, PBFT provides a *view-change* protocol to jointly agree on changing the leader.

However, Miller et al. [3] have shown that PBFT-like algorithms can be halted arbitrarily long if the adversary has arbitrary control over the network and the ability to crash any node at a time. The throughput of weakly synchronous protocols degrades substantially in the presence of unpredictable network behaviour. Moreover, it is often unclear, how to set timeout parameters in a real-world network.

Note that the nature of the PBFT algorithm requires a *quadratic number of message exchanges* (between the participants), i.e. a communication complexity of $\mathcal{O}(n^2)$. Moreover, throughput is limited by the current leader's bandwidth. There have been many proposals to improve PBFT-like protocols, mainly through *optimistic execution*.

PBFT and similar, traditional consensus protocols rely on replication mainly to achieve resilience against faulty (or malicious) nodes, not to achieve a scalable system. In particular, mechanisms such as *sharding* of resources have not been any concern, with a few exceptions (e.g. a work by Gray and Lamport [8]).

B. Non-Byzantine Fault Tolerance

There are several notable algorithms in the field of non-Byzantine fault tolerance. Two of them are Paxos [9] and Raft [10]. While the underlying protocols are similar to PBFT, one key difference to PBFT (and all BFT protocols) is that those non-Byzantine algorithms can only tolerate crash-faults (up to $\frac{1}{2}n$ of the nodes), whereas Byzantine algorithms can tolerate arbitrary (including maliciously) corrupted nodes.

In contrast to Paxos, Raft was designed with the goal of providing an understandable protocol. Moreover, it uses randomisation to improve the leader election.

C. Proof-of-Work

Proof-of-Work (PoW) is a term referring to an algorithm that ensures a provable, randomised, and Sybil-attack-resistant selection of a leader (or block producer) based on partial hash collisions. Moreover, it serves the purpose of defining the main chain of blocks as the chain with the largest cumulative difficulty. The difficulty of a PoW is proportional to the size of the partial hash collision. PoW-based protocols inherently need to rely on incentivisation mechanisms to enforce security [11].

Used in Bitcoin's Nakamoto consensus, PoW is able to overcome the FLP impossibility result. Its gossip protocol does *not* require quadratic communication overhead.

While PoW-based systems are often said to be secure as long as the majority ($> 50\%$) of nodes are honest, most such systems suffer from additional attack vectors such as selfish mining [12] – effectively reducing the amount of malicious actors needed to about 25%.

Probably, the most significant drawback of PoW-based protocols is the immense energy consumption, which does not produce anything of use. This led to a plethora of new consensus protocols being proposed, including more energy efficient replacements such as Proof-of-Stake. Moreover, some proposals have been made to introduce proofs of useful work [13].

1) Mining Pools: Many miners in traditional PoW-based systems join so-called mining pools to reduce the variance of their rewards. However, such mining pools effectively lead to centralisation and thwart transaction censorship resistance, because a malicious pool owner can control which transactions should be mined by his pool.

There have been a few proposals to create decentralised mining pools [14], [15], and Nimiq currently advocates their own such solution.

D. Proof-of-Stake

The term Proof-of-Stake (PoS) generally refers to any system where block proposals are made and voted on by those, who can prove ownership of a stake of coins in the network. PoS is often proposed as an energy-efficient alternative to PoW.

It is common for PoS protocols to elect a leader from within the stakeholders, who is then responsible for appending transactions to the blockchain during a specified epoch of time. Such a leader election may be public (e.g. Ouroboros [16]) or private (e.g. Ouroboros Praos [17]). In a private leader election, each node needs to check whether it will be the next leader using its private information but then can prove to others using only public information that it is indeed the next leader. Such a design makes it impossible for others to predict and carry out DoS attacks against the next leader until it is too late.

Naive implementations of a PoS system are vulnerable to *nothing-at-stake attacks*, i.e. validators can vote in favour of multiple conflicting blocks at the same height. These attacks are usually mitigated by a mechanism known as *slasher* [18]. Validators must not only prove ownership over their stake but must also place a security deposit, which will be burned if the validator is found to propose or vote for conflicting blocks.

Another type of attack is called *grinding attack* and generally describes a scenario, in which a miner or leader repeatedly recreates the next block secretly in order to influence the next leader election. Such an attack is only possible if the leader election depends on input that can be biased by the miner or leader. Hence, mitigation techniques include the use of a unbiasable source of randomness.

Finally, PoS protocols can be susceptible to *long-range attacks*, i.e. a participant can bribe other participants to sell their private keys. Such an attack works if the stake is determined by the historic stake and the bribed participants have already moved out their funds. The sold private keys can then be used to increase the voting power of a malicious participant.

Variants of PoS include:

- **Proof-of-Deposit:** Stake needs to be locked away in order to participate.
- **Proof-of-Burn:** Stake needs to be destroyed in order to participate.
- **Proof-of-Coin-Age:** Stake is weighted by coin-age.

An advantage over PoW is that the cost of attacks in PoS is always proportional to the market capitalisation, whereas in PoW systems, efficient technology enables much cheaper attacks at only the value of the reusable technology.

E. Proof-of-X

Replacements for PoW other than PoS have been proposed as well. The two most notable approaches are Proof-of-Capacity (or Proof-of-Space) and Proof-of-Elapsed-Time.

Examples of Proof-of-Capacity-based protocols are PermaCoin and SpaceMint. Proof-of-Capacity often requires participants to store large amounts of data as part of the proof. Hence, a design challenge for Proof-of-Capacity-based protocols is their resilience to centralisation in the form of outsourcing the storage to a centralised service.

All currently known approaches for Proof-of-Elapsed-Time rely on trusted computing environments such as Intel SGX. Therefore, breaking one such trusted computing environment gives an attacker unlimited power to attack the system.

IV. From Closed to Open Networks

A lot of protocols are designed to work in closed networks only, with participants known upfront. Thus, to employ such a protocol in a non-permissioned, open blockchain, it is necessary to choose a subset of all participants that then is closed and well-defined. Such a subset is often selected for a limited time only, called epoch. Prominent names for such a subset are *validator set* or *committee*. After each epoch, the committee can either change wholly or partially. Moreover, different criteria can be applied to select the new committee members.

A. PoW-based Selection

The most straight-forward selection criterion is probably PoW. The participant who can solve a PoW puzzle first will replace the oldest member of the previous committee. Such an approach results in a rolling window committee, replacing a single member each epoch. ByzCoin [19] is an example protocol that uses this leader selection algorithm.

One issue that arises from PoW-based selection is leader contention, i.e. multiple protocol participants solve the PoW puzzle at the same time.

B. Public Randomness-based Selection

Other selection criteria include verifiable randomness or cryptographic sortition (see Section XVIII-B2) protocols. Such algorithms often take previous blocks as input to generate randomness or alternatively rely on multi-party coin-tossing protocols. They can be used either to replace the whole committee at once, or to replace a subset of the oldest members of the committee. Algorand [20] and Snow-White [21] are examples for such protocols replacing the full committee each epoch. Omniledger [22] is an example that only replaces a subset of the committee.

For such protocols, it is essential to prevent malicious committee members from being able to influence the result of the next committee selection.

V. Protocols

In the following sections of this paper, we will discuss a selection of consensus protocols representative for the vast space. We will categorise the protocols into seven different areas, each focusing on different protocol aspects.

1) **PoW-based blockchains:** This area includes examples of simple PoW-based blockchains and improvements.
2) **PoS-based blockchains:** This area focuses on PoS-based blockchain protocols.
3) **Hybrid protocols:** This area focuses on protocols combining traditional consensus protocols and PoW- or PoS-based solutions.
4) **DAG-based protocols:** This area explores protocols that extend the simple notion of a blockchain to directed acyclic graphs.
5) **Separate Account Chains:** This area discusses protocols that aim to create a single chain per account.
6) **Leaderless protocols:** This area includes protocols that do not require a leader or designated committee, some of them being simple closed-network BFT protocols.
7) **Sharding:** Lastly, we look at protocols that promote sharding in blockchains.

It should be noted that this is just one way to organise those protocols.

Due to the large number of protocols, we put each of those areas into their own section.

VI. PoW-based Blockchains

Bitcoin is the first and still most prominent example of a PoW-based blockchain. Hence, much work has been carried out in proposing possible improvements to Bitcoin-like blockchains. In this section, we will briefly introduce Bitcoin and Bitcoin-NG. Bitcoin-NG is an academic approach to improve Bitcoin's scalability and the basis for many other protocols.

A. Bitcoin's Nakamoto Consensus

Bitcoin [23] uses a Proof-of-Work-based consensus algorithm, which only provides probabilistic guarantees and no finalisation. As mentioned before, one of its key advantages over PBFT-based algorithms is that it does not require a quadratic communication overhead, but instead relies on a gossip protocol. Moreover, it works in open, permissionless systems without modifications. Due to the choice of PoW, Bitcoin is only able to offer weak consistency.

Several protocol-level attacks (e.g. selfish mining [12]) exist that lower the effective security threshold of Bitcoin to well below 50% (about 25%).

B. Bitcoin-NG

Bitcoin-NG [24] relies on the same security and trust model as Bitcoin but improves performance by separating leader election from transaction serialisation. To this end, Bitcoin-NG distinguishes between *key blocks* and *micro blocks*.

Key blocks are similar to regular Bitcoin blocks (just without any transactions) and are used to select the current leader. Then, once the leader is chosen, it is entitled to serialise transactions unilaterally into micro blocks. Micro blocks can be much more frequent than key blocks.

In order to correctly align the incentives of the next key block miner to build upon the latest micro block, transaction fees are shared between both the current micro block producer (40%) and the next leader (60%). The authors also justify the choice of this split.

Since micro blocks are not secured by PoW, they are cheap to fork and thus could lead to double-spending attacks. This is disincentivised by allowing others to report such behaviour

using a *poison transaction*, which invalidates the revenue of a fraudulent leader.

Temporary forks due to multiple key blocks being mined at the same instant can cause the system (similar to Bitcoin) to be in an inconsistent state until the fork is properly resolved. This can take 10 minutes or even more.

VII. PoS-based Blockchains

Ouroboros [16], Ouroboros Praos [17], and Snow-White [21] all are PoS-based protocols. One of the most interesting differences between them is their leader election process.

In Ouroboros, a subset of all stakeholders first runs a multi-party coin-tossing protocol to agree on a random seed, which is then fed into a pseudo-random function that elects a leader from among them weighted by the stake. The same random seed is also used to determine the subset of stakeholders for the next epoch. Rewards are distributed among all members of the subset. This is an example of a public leader election process. Ouroboros requires most participants of the coin-tossing protocol to be incorruptible by an adversary for the whole epoch.

Ouroboros Praos and Snow-White employ a private leader election process. While Snow-White relies on a PoW and a weakly synchronised clock, Ouroboros Praos uses a verifiable random function (similar to Algorand [20], see Section VIII-D).

VIII. Hybrid Protocols

This section discusses protocols that are inspired by classic BFT consensus systems, such as PBFT [4], combining such classic protocols with PoS or PoW into a hybrid protocol.

In particular, we will start with Tendermint, a blockchain protocol bringing PBFT to the open, permissionless setting. Among other protocols, we also introduce ByzCoin, a protocol that dramatically improves upon many of PBFTs caveats in an open blockchain. Finally, we take a look at Algorand, a protocol involving public randomness.

A. Tendermint

Tendermint is a consensus protocol proposed in 2016 [25]. It closely follows the PBFT model but is designed to be more practical. The consensus algorithm can be divided into the following components:

1) **Proposals:** Each new block is proposed by the current leader, which changes between rounds (roughly corresponds to PBFT *pre-prepare*).
2) **Votes:** Two phases of voting called *pre-vote* and *pre-commit* (corresponds to PBFT *prepare* and *commit*), before a block is called a *commit*.
3) **Locks:** This is an additional mechanism, preventing nodes (validators) to commit to more than one block at the same height. It is crucial that such a locking mechanism is implemented in a way that does not compromise liveness.

One difference to PBFT is that the leader is rotated in every round, which makes skipping a faulty leader much easier and reduces the communication overhead in this case significantly (there is no need for running a leader election protocol since the new leader can be determined deterministically from the validator set).

In order to determine and change the closed set of validator nodes, Tendermint takes an approach similar to Raft [10]. Validator set changes must pass through consensus.

Tendermint provides full accountability for validators, i.e. they can be proven to have acted maliciously. If there is a proof of misbehaviour, a validator can be punished. This can be achieved through a PoS protocol.

The communication complexity of Tendermint is $\mathcal{O}(n^2)$.

B. Casper

Casper [26] is a generalisation of Tendermint. It has been proposed as a possible PoS protocol for Ethereum (which currently still employs a PoW consensus).

C. ByzCoin

ByzCoin [19] is a PBFT-based protocol ported to an open setting. PoW is used to determine a rolling window validator set, with the leader being the latest PoW winner. ByzCoin also adapts Bitcoin-NGs [24] concept of key blocks and micro blocks but considers them as two separate chains. The key block chain is used to determine the validator set and leader via PoW. Both chains are secured using PBFT, but only the micro block chain contains the transactions.

The authors of ByzCoin dramatically improve upon a naive PBFT implementation in four key areas:

1) **Open membership:** ByzCoin uses the PoW from key blocks to determine the validator set (e.g. 144 last block producers), giving recent miners voting power proportional to their recent hash power.
2) **Scalability to hundreds of validators:** While PBFT is only designed for small environments (typically not more than 16 replicas), ByzCoin uses CoSi [27] (communication tree-based collective Schnorr signatures) in order to reduce the costs of PBFT rounds and also the costs for light clients to verify a transaction commitment. Light clients must only verify a single, collective signature ($\mathcal{O}(1)$) instead of the individual signatures of the super-majority of validators ($\mathcal{O}(n)$).
3) **Transaction commitment rate:** Like Bitcoin-NG, ByzCoin relies on separating leader election (key blocks) from transaction ordering (micro blocks) to achieve higher throughput. However, Bitcoin-NG uses a single blockchain for both type of blocks, which creates a race condition between the latest micro block and the next key block. Hence, ByzCoin separates both types into their own blockchains.
4) **PoW block conflicts:** If two key blocks are mined simultaneously, a high-entropy, deterministic prioritisation function is used to break the tie. By using all competing blocks' header hashes as an input, selfish mining can be

mitigated. However, PoW block conflicts can still lead to a temporary interruption of liveness: if one part of the validators already committed to one of the blocks, while another part committed to a different block, consensus can stall until the next key block is found.

Usage of CoSi signatures also reduces the communication complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(\log n)$, since validators only need to communicate along the tree. However, practical issues apply if some validators are offline during the collective signing process (which takes two rounds, just as in Schnorr multisignatures): the CoSi protocol can still proceed, but the size of the resulting signature needs to grow in order to accommodate metadata documenting who did participate and who did not. If the leader suspects liveness of the CoSi scheme being compromised, it can revert to a flat $\mathcal{O}(n)$ collective signature scheme, without involving a tree-based optimisation.

ByzCoin forms a hash power-proportionate committee and can achieve a near-optimal tolerance of f faulty committee members among $3f + 2$ in total. Their evaluation suggests a throughput of around 1000 txs/sec, offering strong consistency.

ByzCoin – like many BFT protocols – is vulnerable to slowdown or temporary DoS attacks that Byzantine nodes can trigger. A Byzantine leader might *temporarily* exclude minorities ($< \frac{1}{3}$) from the consensus process.

In contrast to Bitcoin, ByzCoin is less prone to eclipse attacks, since an attacker cannot convince a victim into accepting an alternate attacker-controlled transaction history. In order to carry out such an attack, the attacker would need to control two-thirds of the voting power at some point of the chain in the first place.

In contrast to ByzCoin, Tendermint – which also implements a PBFT-like algorithm – only considers small validator sets (up to 64) and does not address link-bandwidth between the validators.

D. Algorand

Algorand [20] uses a novel Byzantine Agreement (BA) protocol and Verifiable Random Functions (VRF) for the committee election. It guarantees that the probability for forks is negligible, which is a significant advantage over classical PoS systems as it mitigates attacks such as the nothing-at-stake problem. The design of their protocol is such that it allows scaling to hundreds of thousands of participants.

A key feature of Algorand is that no participant needs to keep private state. Although the BA protocol employs multiple rounds, the committee is reelected for every such round and a committee member always only performs a single vote. Participants need to find out in private whether they were elected and then publish a single message in their term as a committee member. This feature makes Algorand committee members resistant against targeted DoS attacks since the committee members are only then known to the attacker, when they have already served and published their vote. Other hybrid consensus designs are only secure with respect to a mildly adaptive adversary that needs more time to compromise the committee than the epoch they serve in.

The BA protocol does not require leaders (except for the block proposal). To prevent Sybil attacks, Algorand weighs participants by their stake in the system (similar to PoS). Cryptographic sortition allows electing the committee based on these weights. Weighing by stake implicates the security requirement that $\frac{2}{3}$ of all funds belong to honest users. Conversely, an attacker must invest substantial financial resources, before a successful attack can be carried out.

To achieve liveness, Algorand assumes strong synchrony, whereas safety only requires weak synchrony. Moreover, Algorand assumes loosely synchronised clocks across all users. Thus it requires stricter assumptions on the network than other solutions. Communication in Algorand is gossip based, resulting in a similar communication complexity as is the case for Bitcoin.

Blocks in Algorand can either reach *tentative* consensus or *final* consensus. A block with final consensus will also transitively finalise the consensus for previous blocks with tentative consensus (since final blocks are totally ordered). The BA algorithm may declare tentative consensus, if it cannot confirm the absence of other blocks, e.g. due to network asynchrony. Blocks are proposed by a block proposer, who is chosen as the participant which can prove to have the lowest random value output by his VRF. In practice, the gossip protocol ensures that only those blocks with a proof for the lowest value are actually propagated throughout the network.

Committees (that are changing in every BA step) then try to agree on a block. This is done in two phases. The first phase reduces agreement on any block to the problem of agreeing to one of two options: agreeing on the empty block or on a single proposed block. The second phase then serves the purpose of agreeing on one of these two options. In each step, committee members cast a vote for a value, and every participant (not restricted to the committee) counts the votes. Algorand requires users to keep track of potential forks and resolve them using a recovery protocol in regular intervals using a fork proposer.

The input to the randomness primitives is a random seed, which is provably generated using the block proposer's VRF from the previous seed. This requires bootstrapping the system with a start seed, ideally chosen through distributed random number generation.

Algorand can achieve up to 875 txs/sec, in comparison with Bitcoin achieving 7 txs/sec.

Limitations of the Algorand protocol (left for future work) include missing incentive systems and missing forward security. Since Algorand certifies the consensus to new users, an attacker managing to obtain enough user keys could create a fake certificate and thus a fork.

E. Seagull

Seagull is a proposal by Franca et al. from Trinkler Software for their blockchain-based financial system called Katallassos. It represents a mix between PoS and PBFT [4] inspired by ByzCoin [19].

In this protocols, validators need to signal their interest by locking collateral. At any point in time, they can withdraw their interest and unlock their collateral after a cool down period. For practical reasons, they allow validators to choose a different signing key than the one corresponding to their account.

The validator set is fully replaced in every key block (instead of a rolling window as in ByzCoin). It is sampled from a random seed generated using the same scheme as Ouroboros. The micro blocks between two key blocks are therefore divided into three phases with recurring leaders.

1) To generate the random seed for the next round, each micro block leader, in the first phase, needs to create a random value. A cryptographic commitment to this value is put into the micro block. Moreover, a verifiable secret sharing scheme is used to create shares of this value, such that a threshold of those can be used to recover the value. Each share is encrypted for one of the other validators and put into the block as well.
2) In the second phase, honest validators will reveal their commitment.
3) In the third phase, honest validators can use their encrypted shares to reconstruct the random values from such validators that have not opened their commitment.

Then, all random values will be incorporated into the entropy for the next round.

Since Katallassos relies on an accounts tree, syncing the blockchain can be easily done by downloading all key blocks, the accounts tree, and only the micro blocks since the latest key block.

IX. DAG-based Protocols

Another whole category of distributed ledger protocols are those based on directed acyclic graphs. Strictly speaking, these are not blockchain protocols, but rather block-graph protocols.

GHOST is one such protocol that is still very close to Bitcoin, as it uses the graph only to determine the correct main chain. The Inclusive Blockchain Protocol is another example of such a protocol. Finally, we will introduce Spectre, which does not require the notion of the main chain any more.

A. GHOST

The GHOST protocol [28] describes a modification of Bitcoin's chain selection algorithm. Its goal is to improve resilience and scalability of Bitcoin by including blocks that are not on the main chain into the chain selection. Instead of choosing the chain with most cumulative work, this algorithm chooses the heaviest chain as the main chain. The weight of a block is calculated from the size or cumulative work of its subtree.

The main limitation of this approach is that the full tree needs to be available to all nodes. Also propagating those blocks outside the main chain, however, makes the system much more vulnerable to DoS attacks.

B. Inclusive Blockchain Protocol

The Inclusive Blockchain Protocol [29] proposes the use of block-DAGs. In contrast to the next protocol, Spectre [30], it still relies on a simple chain-based protocol to provide a total order over blocks. The authors provide a comprehensive game-theoretic analysis of their protocol.

C. Spectre

Spectre [30] is a PoW-based protocol that allows miners to mine blocks concurrently and structures those blocks in a block-DAG instead of a linear blockchain. A key ingredient of Spectre is loosening the properties in comparison to what classical consensus requires. Instead of requiring that the order between transactions must be decided and agreed upon by all honest participants, Spectre requires this only for transactions performed by honest users. Spectre can guarantee that security in the presence of double spending but may delay the decision regarding double spends arbitrarily.

By maintaining a full DAG, miners can create blocks more frequently and concurrently. There is no need to agree on a single main chain, nor do mining nodes need to know the propagation delay in the network. More frequent blocks do also benefit the decentralisation, since rewards will be much more frequent, reducing the need for mining pools.

The block creation and communication instructions in Spectre are simple: When a new block is received or created, it is gossiped to all peers. When creating a new block, a list containing the hash of all leaf blocks in the locally-observed DAG is embedded (this list can be limited in size).

In order to verify which and whether transactions have been accepted, Spectre defines a pairwise relation $x \prec y$ over the blocks (informally, they say x *precedes* or *defeats* y). Using this relation, nodes can identify accepted transactions and quantify how robust the acceptance is (i.e. a bound on the finality). It is important to note that the relation is not transitive. The relation coincides with Bitcoin's longest-chain rule in the case of a simple fork between two chains. It is able to prevent censorship and double spend attacks.

A transaction is accepted with certain robustness if the block containing it robustly precedes (or defeats) all counterparts (and conflicting transactions). The relation has the property that, once a block is published, the set of blocks that precede it in the pairwise ordering closes fast. With high probability, it only consists of blocks that were published before or right after its own publication. Since a transaction might occur in multiple blocks, Spectre defines equivalence classes on transactions, also splitting the actual transaction from the fee part.

If two blocks include the same transaction, the fee portion of the transaction constitutes a double spend. To resolve these kinds of double spends, Spectre proposes the use of so-called *settlement transactions*. The different block producers can voluntarily sign such a transaction in order to evenly split the transaction fee and resolve the double spend.

Note that it also is considerably more complex to implement a difficulty adjustment in a DAG. Spectre ensures that only a single block is responsible for the difficulty adjustment at

regular intervals. However, due to the nature of a DAG, one must also accept blocks with an outdated target around the time of the adjustment, which can be determined by graph properties.

A significant drawback of Spectre is that the calculations used for the block relation are not inherently efficient.

X. Separate Account Chains

Other recent protocols, such as Nano or Blink, propose to parallelise and scale blockchains by splitting a single blockchain into many. In particular, they consider a single blockchain per account. However, since transactions always interact with two accounts, care has to be taken about when and how to update each account chain.

A. Nano

Nano [31] is a block-lattice based protocol, in which each account has its own blockchain. It is built on top of UDP and only uses TCP for the bootstrapping process.

Each account chain can only be modified by the owner and contains different types of blocks. There are *open*, *send*, *receive* and *change* blocks. Nano's full supply is stored in a *genesis account*. Their whitepaper does not make clear who controls this account.

Transactions are split into a sender and a receiver part so that the receiving account has full control over the order in which incoming transactions are processed. The sender has to incorporate a *send* block into his account chain (which is considered immutable, once confirmed by a quorum), the receiver a corresponding *receive* block. Following from this separation, transactions can be settled and unsettled. Settled transactions are such that the receiver has already generated a receive block, unsettled transactions are not yet part of the receiver's account balance. Transactions include a PoW field, which is solely used as an anti-spam measure.

Each account owner must choose a representative that can vote on its behalf (which can be the owner itself). The representative can also be changed using a *change* block at any time. The only scenario in which actual voting occurs is upon detection of a fork. Representatives vote on one of the forked blocks. Their votes are weighted by the sum of balances of all accounts that have chosen this representative. The whitepaper does not go into detail about what happens if both receivers of a double-spend try redeeming the transactions or how a roll-back is implemented.

Nano plans on implementing *block cementing* to prevent blocks from being rolled back at a certain point in time.

B. Blink

Similarly to Nano, in Blink, every account has its independent chain of transactions. These independent chains are managed by an account supervisor (or *locker*) that is chosen using a PoS type of election. The protocol is divided into rounds, and each transaction needs to be assigned a round. Blink assumes a synchronous network and reduces communication complexity by a gossip algorithm.

Besides the balance, an account also holds a data field and a chained transaction hash (CTH). The CTH essentially is a hash over the address and the ordered list of transactions, recursively computed as a chain of hashes over the old states. Moreover, the CTH also contains the hash of the global state at a previous moment in order to make past transactions irreversible: Modifying any transaction before that point would invalidate all subsequent transactions.

Transactions are only valid for given CTHs of sender and recipient and a given round id. Thus, if there are two incoming transactions with the same recipient CTH, only one can be applied.

An account can have more than one locker, and lockers are chosen using a weighted random selection according to their stake. Lockers have to lock away collateral. Since lockers can be randomly reassigned at the beginning of each round, lockers have to either keep the global state or sync the state of the respective accounts. A proposed optimisation could swap only a subset of lockers.

Locker selection is carried out via a weighted function. This function takes as an input an accounts address, the global state hash, and the round id. It hashes the input repeatedly until the resulting hash is not considered biased anymore. It is unclear whether this approach is sufficient to remove any bias in locker selection. While the whitepaper does not compare this approach to a VRF, it seems reasonable that this approach could be replaced by relying on such a VRF.

A double-spend can only occur if at least one locker signed two conflicting transactions. In this case, both transactions will be rejected, and a punishment transaction will take funds from the locker's collateral and distribute them. In order to prevent conflicting transactions across rounds, it is proposed that transactions need to be signed by lockers from the current and previous round.

Blink's global state consists of all participating nodes (i.e. those who locked away collateral to be considered as a locker), all accounts, and all system variables. The accounts are stored in a Merkle Radix Tree, like for Ethereum. Having the system variables as part of the global state allows to vote and modify those as part of the protocol.

Blink's consensus protocol consists of two phases: (1) proposal and sync, and (2) commitment. During the first phase, every node computes a Merkle hash over all transactions of that round (in chunks). This way, two nodes can easily find differences in their states and sync the relevant transactions. In the second phase, every node can propose only once the final set of transactions for that round. Committing to more than one value will be punished to prevent nothing-at-stake attacks.

The authors also propose to employ sharding but do not give any details.

XI. Leaderless Protocols

This section discusses protocols that do not require a leader or designated committee, either in a closed or in an open network. The most important protocols presented

here are HoneyBadgerBFT, Ouroboros-BFT, and Stellar. While HoneyBadgerBFT and Ouroboros-BFT are only applicable to permissioned, closed networks without modification, Stellar is designed for an open setting.

HoneyBadgerBFT is a protocol providing Byzantine fault tolerance in fully asynchronous networks, by relying on recent cryptographic techniques and randomisation to overcome FLP impossibility. Ouroboros-BFT is a BFT protocol inspired by the PoS blockchain Ouroboros. Stellar is a federated Byzantine agreement protocol that relies on configurable trust assumptions to guarantee safety.

A. HoneyBadgerBFT

The authors of HoneyBadgerBFT [3] argue that timing-assumptions made by PBFT-like algorithms are ill-suited for cryptocurrency deployment scenarios. Their main argument is that an adversary with arbitrary control over the network and the ability to crash any node at a time can cause PBFT to halt for arbitrarily long. They note that such an attack is not specific to PBFT, but fundamental to any protocol that relies on timeouts to cope with crashes.

Therefore, they introduce the first practical consensus algorithm that does not require timing assumptions. They significantly improve over prior work, by reducing redundant work among nodes and instantiating building blocks with more recent cryptographic primitives.

Similar to PBFT, HoneyBadgerBFT is designed for closed and well-defined validator sets only. While, according to the authors, their protocol could provide a better throughput than the classical PBFT protocol, using their protocol in a non-permissioned blockchain scenario requires a trusted party for the initial setup and for validator set changes. Moreover, HoneyBadgerBFT relies on relatively new crypto assumptions. The authors state that in a real deployment, the trusted party could be replaced by a distributed key generation protocol (c.f. Boldyreva [32]).

The HoneyBadgerBFT protocol roughly works as follows. In the setup phase, the trusted party employs a threshold encryption scheme that allows any party to encrypt a message to a master public key. To decrypt such an encrypted message, $f+1$ nodes can compute and reveal decryption shares that can be used to recover the original message (f being the number of faulty nodes). The trusted party creates the master public key and the secret key shares and hands them to the validators in the given validator set. Now, for each epoch:

1) Each validator chooses a random set of transactions (to minimise redundant transactions and thus redundant work) and encrypts this set using the threshold encryption.
2) All validators pass their encrypted set to an asynchronous common subset protocol, which is used to agree on a set of ciphertexts.
3) Collectively decrypt the ciphertexts previously agreed upon, sort the union over the transactions contained and append them to a block.

Using a threshold encryption scheme prevents a malicious party from choosing only such transaction sets that do **not** include a specific transaction in step 2). Removing the threshold encryption would thus result in violating censorship resilience.

In their evaluation, they achieve a throughput of more than 20,000 tx/sec for networks of up to 40 nodes and more than 1,500 tx/sec for networks of 104 nodes.

B. Ouroboros-BFT

Ouroboros-BFT [33] is a protocol designed by IOHK. It is inspired by Ouroboros [16], a PoS blockchain protocol. Ouroboros-BFT offers transaction processing at full network speed, instant confirmations, and instant proofs of settlement. It is shown to be resilient against $\frac{1}{3}n$ malicious nodes in a synchronous network and $\frac{1}{2}n$ malicious nodes in the covert adversarial model, which may be enforced through penalty mechanisms. Ouroboros-BFT assumes a globally synchronised clock and shows how it can be simulated in a synchronous network.

The protocol is designed to be simple: It does not rely on multiple phases. Time is divided into separate, discrete slots $sl_1, sl_2, \ldots$. Ouroboros-BFT assumes a closed, well-defined set of participants. In each time slot, participants execute the following three steps:

1) **Mempool update:** Add new, consistent transactions to the mempool.
2) **Blockchain update:** If aware of a longer, valid blockchain, adapt this new blockchain.
3) **Blockchain extension:** Blockchain production works round-robin according to the time slots. If the current participant is responsible for the next block, create a new block containing all valid mempool transactions. Apart from the transactions, the block contains the timeslot, a signature of the timeslot, the hash of the previous block, and a signature of the block.

Blocks are considered finalised if their slot time is more than $3t + 1$ in the past. Other blocks are considered pending. Transactions acquire their final sequence in the ledger after $5t + 2$ ticks of the global clock, with t being the number of malicious nodes.

For blockchain synchronisation, it is sufficient to download a set of recent blocks and searching for a segment called a *dense witness*.

In order to run Ouroboros-BFT in a permissionless network, slots can be assigned to epochs, and the set of validators may change between epochs.

Compared to PBFT [4], Ouroboros-BFT speculatively processes all transactions, while transaction serialisation is done lazily. Communication complexity is reduced to $\mathcal{O}(n)$ in the optimistic case and $\mathcal{O}(nt)$ in the worst case, compared to $\mathcal{O}(n^2)$ for PBFT. Ouroboros-BFT assumes a synchronous network for guaranteeing safety, while PBFT provides safety with unbounded network delays. It has to be noted that an asynchronous network can lead to temporary forks in Ouroboros-BFT, which however will be resolved as soon as the network returns to synchrony.

C. Stellar

Stellar [34] uses a federated Byzantine agreement protocol. Thus, each participant has to define a set of other participants it trusts. This makes the trust model nontrivial, requiring a correct configuration of trustees by each participant.

The protocol relies on so-called *quorum slices*. Let a *quorum* be a set of nodes sufficiently large to reach agreement; then a quorum slice is a subset of the quorum convincing one particular participant of agreement.

The consensus protocol votes on statements in multiple phases: In a first phase, nodes accept a statement under predefined conditions (e.g. by a quorum). In a second phase, nodes confirm the statement. The protocol is inspired by ballot-based approaches such as Paxos [9].

Their main blockchain protocol works by nodes nominating sets of transactions and the network agreeing and confirming upon one such set. In order for the nomination protocol to converge, nodes are assigned a temporary priority, based on a *slot* specific hash. A slot is a concept similar to a block, referring to consecutively numbered updates in a sequentially applied log.

The Stellar consensus protocol can only guarantee safety when nodes choose adequate quorum slices and thus depends upon a user-configurable parameter. Like other BFT protocols, Stellar can suffer *perpetual preemption*, meaning that reordering or delaying of messages can arbitrarily delay the consensus. Additionally, if non-faulty nodes were tricked into bad commit messages, the only way to unblock their local consensus is by rejoining under a new identity. Stellar requires continuity of participants over time since nodes need to rely on their trustees.

D. Ripple

Ripple [35] implements a low-latency PBFT variant that is based on collectively-trusted subnetworks and infrequent membership changes. Ripple publishes a membership list that participants can edit for themselves. It has been shown, however, that divergent lists invalidate safety guarantees.

XII. Sharding

While replacing Nakamoto consensus with PBFT [4] already increases throughput and decreases commit latency, still all validators have to redundantly validate and process all transactions. An increase in participants thus means a gradual decrease in performance due to the coordination overhead incurred.

Sharding splits transactions across multiple committees and thus can scale with the number of validators. Prominent examples of sharded protocols are RSCoin [36], Elastico [37], and OmniLedger [22].

OmniLedger is a solution for sharding based on a ByzCoin-like protocol. While OmniLedger has a flat committee topology, Elastico's topology is hierarchical. However, Elastico introduces new security assumptions and trades performance for security. RSCoin uses a central bank and miners that are authorised by this bank and hence forfeits permissionless decentralisation.

In general, sharding permissionless blockchains in a Byzantine setting is challenging and only a few protocols exist aiming to solve this problem.

A. Elastico

Elastico [37] explores sharding in a permissionless setting. It uses the least-significant bits of the PoW hash to distribute miners to shards and runs PBFT within each such shard to reach consensus. A leader shard then verifies all signatures and creates a global block. Elastico has a high DoS resistance since the committee is always entirely swapped.

However, Elastico's shard selection is not bias resistant, as PoW miners can influence the outcome of the shard allocation. Moreover, its relatively small shards are prone to high failure-probabilities: for 16 shards, the failure probability is 97% over 6 epochs. Elastico also does not ensure atomicity for cross-shard transactions, potentially leaving funds locked forever. Finally, validators constantly switch shards and thus need to store the global state.

B. OmniLedger (and ByzCoinX)

OmniLedger [22] is a protocol designed explicitly to scale-out, i.e. to grow the processing capacity with the number of validators. To this end, OmniLedger relies on sharding, choosing the shards with a bias-resistant public randomness protocol. It also introduces an atomic cross-shard commit protocol. OmniLedger can support Visa-level workloads, confirming transactions in under two seconds. It builds on Ouroboros [16], Algorand [20], Elastico [37], and ByzCoin [19].

OmniLedger separates identity management from transactions and thus runs a separate identity blockchain for validators to establish their identity and interest to serve in a committee in the next epoch.

This identity blockchain, as well as the shard assignment, assume strong synchrony, whereas all protocols inside one epoch only rely on a partial synchrony assumption.

OmniLedger takes great care to not compromise security or permissionless decentralisation (in contrast to other solutions). Thus, it solves the following key challenges:

1) choosing statistically representative groups of validators via permissionless, Sybil-attack-resistant algorithms such as PoW or PoS,
2) guaranteeing a negligible probability that any shard is compromised by forming sufficiently large and bias-resistant shards,
3) handling cross-shard transactions correctly and atomically.

While their protocol, in principle, can handle up to $\frac{1}{3}$ of malicious validators, they only describe parameters optimised for up to 25% of malicious validators.

OmniLedger runs a public randomness or cryptographic sortition protocol within the prior validator set in order to choose the next validator set according to the stake distribution. OmniLedger achieves strong bias resistance by using

RandHound [38], a scalable, secure multi-party computation protocol that provides unbiasable, decentralised randomness in a Byzantine environment. The use of RandHound can solve both the first and second key challenge.

In order to run RandHound for the shard distribution, a leader is required to coordinate the protocol. This leader is chosen similarly to Algorand using a VRF. OmniLedger does not swap out the validators of a shard at once but swaps them gradually. This enables a continuous service by the remaining (non-swapped) operators in a shard. Note that the shard distribution protocol (including RandHound) might take up to approximately 3 hours using 1800 participants. Thus, the length of an epoch is chosen to be one day.

The third key challenge, cross-shard transactions, is handled by a new two-phase lock/unlock protocol called Atomix, coordinated by a client. OmniLedger shows how to implement Atomix for either UTXO- or state-based models. In the UTXO model, the client first submits the transaction to the input shards, which provide either a proof-of-acceptance or a proof-of-rejection, locking the funds. A single proof-of-rejection can be used to unlock the funds in any input shard. The transaction can be completed in the output shards if all input shards provided a proof-of-acceptance.

Periodic, consistent state blocks allow clients to easily sync the current state of a shard without verifying the entire history. These state blocks are produced by classical distributed checkpointing principles [39]. The concept of state blocks ensures that validators do not need to store the full transaction history and helps to switch between shards.

OmniLedger also introduces ByzCoinX, an improved version of ByzCoin [19]. In particular, it enhances ByzCoin to preserve performance under DoS attacks by adopting a more robust group communication pattern. Specifically, it replaces ByzCoin's communication tree used in CoSi by a two-level tree, providing better robustness.

Finally, OmniLedger proposes ByzCoinX to make use of a block-DAG similar to the one proposed in the Inclusive Blockchain Protocol [29]. The authors observe that conflicts between transactions can only arise if two transactions try to spend the same input or if two transactions depend on each other. Hence, the commitment of other transactions can be parallelised safely in a block-DAG. OmniLedger also introduces a two-level approach for transaction verification to further decrease latency.

XIII. Committees

While the use of committees can offer significant performance benefits, there are also drawbacks that need to be considered. Namely, committee members are expected to remain available online during their term. Also, most protocols require direct communication channels between committee members. Serving in a committee can also thwart any anonymity property of a blockchain. Finally, when designing protocols based on committees, DoS resistance against committee members must be considered to ensure liveness.

XIV. Off-Chain Scalability Solutions

In this section, we discuss off-chain scalability solutions such as the Lightning Network.

A. Payment Channels

Payment channels aim to establish secure, two-party ledgers that are kept off-chain but can be enforced on-chain at any point. They are generally limited to work between two predefined parties only and need to lock a certain balance for this purpose. Hence, payment channel networks have been designed to overcome this problem, allowing to perform payments even between parties that are not immediately connected. Such networks use a chain of payment channels and thus need to route payments through a network. REVIVE [40] shows how locked balances in such a network may be rebalanced without on-chain transactions.

However, most payment channel networks still have four fundamental disadvantages:

1) they require complex routing topologies,
2) since funds are allocated to a two-party payment channel, any transfer between non-connected parties will involve fees along the route,
3) payment channels provide only limited privacy guarantees,
4) and most rely on always-online observers to detect and punish misbehaviour.

There are several types of payment channel networks. For example, duplex payment channel networks [41] rely on timelock functionality, while the Lightning Network [42] relies on punishments to enforce honest behaviour. In Lightning Network, if a malicious transaction is detected, honest observers are able to claim all funds of the respective channel.

There is also a Lightning Network equivalent for Ethereum called Raiden network. A competitor of Raiden on Ethereum is Sprites [43], aiming to minimise the worst-case collateral costs of indirect payments.

B. Liquidity Network

Liquidity Network proposes a novel structure for payment channel networks, incorporating off-chain payment hubs. Using Liquidity Network, a user does not require an always-online observer. Moreover, it does not require additional locked funds for additional connections and significantly reduces routing costs. In order to promote decentralisation, they propose that multiple hubs should exist that can be interconnected.

XV. Privacy

In this section, we discuss the privacy-preserving Mimblewimble protocol as well as confidential transactions and BulletProofs. While all those primitives and protocols are mainly targeted towards privacy, they can also enable better (or worse) scalability. Using zero-knowledge proofs and commitments to hide amounts can help aggregating values and

thus reducing the size of the blockchain dramatically. Privacy-preserving blockchain protocols can reduce the publicised information to the minimum needed for validation and ordering of transactions.

A. Confidential Transactions

The notion of *confidential transactions* was introduced by Gregory Maxwell and describes transactions for which input and output amounts are hidden in Pedersen commitments [44]. In order to enable public verifiability, such transactions also carry a zero-knowledge proof asserting that the sum of inputs is greater than the sum of outputs and that all outputs are positive (in the interval $[0, 2^n]$ with n being much smaller than the group size).

Current implementations use range proofs over the committed values, with the proof size being linear in n. Concretely, this can lead to a transaction size of 5.4 KB for a confidential transaction with two outputs and 32 bits of precision. 5 KB of that size is allocated to the range proof.

A key feature of Pedersen commitments is that these commitments work over prime groups and thus can be easily used in arithmetic operations.

B. Mimblewimble

Mimblewimble [45], [46] is a recently proposed protocol that improves upon confidential transactions. It is based on the realisation of Jedusor [45] that the difference between outputs, inputs and transaction fee must exactly yield 0, and that a Pedersen commitment to such a 0 can be viewed as an ECDSA public key (with the private key known to the constructor of the commitment). Thus, when constructing a confidential transaction, a prover can sign the transaction, with the difference of outputs, inputs and fee being the public key.

Poelstra [46] further improved Mimblewimble and showed how to compress the blockchain further, so that it only consists of a small set of block headers, the remaining unspent transaction outputs and corresponding range proofs, and an unprunable 32 bytes per transaction.

C. SNARKs, STARKs and Bulletproofs

Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (SNARKs) [47], [48] are succinct zero-knowledge proofs that, however, require a trusted setup. SNARKs provide constant-size proofs that are extremely fast to verify. They rely on strong unfalsifiable assumptions.

STARKs [49] are zero-knowledge proofs that only rely on collision resistant hash functions. Their proof size grows logarithmically but is much larger compared to SNARKs and Bulletproofs. For example, STARKs require a proof size of 200KB for circuits of size 2^{17} at 60 bit security, whereas Bulletproofs only require 1 KB for 120 bit security.

Bulletproofs [50] are a new non-interactive zero-knowledge proof protocol providing very short (logarithmic) proofs without the need for a trusted setup. Bulletproofs are not only well-suited for efficient range proofs on committed values but can also be used for general arithmetic circuits. Moreover, Bulletproofs support aggregation of range proofs, proving that m commitments lie in a given range. Aggregation is done via a multiparty computation.

Using Bulletproofs, a Mimblewimble blockchain would only grow with the number of transactions that have unspent outputs instead of the size of the UTXO set.

In terms of size, SNARKs are the shortest, followed by Bulletproofs, and lastly STARKs. In terms of verification, SNARKs are the fastest, followed by STARKs, and lastly Bulletproofs. In terms of generating the proof, STARKs are the fastest, closely followed by SNARKs, and lastly Bulletproofs. Only STARKs are considered post-quantum secure.

D. Zcash

Zcash is probably the most established protocol relying on SNARKs. In Zcash, zero-knowledge proofs are used to prove that a transaction is spending a previously unspent coin without publicly revealing which one. This has the considerable disadvantage that it is impossible to prune past transaction.

XVI. Related Work

Bano et al. [51] present in their SoK paper an excellent classification and comparison of different consensus protocols. Focusing on the high-level comparison and systematisation, the authors do not go into depth regarding how these consensus protocols function. In contrast, our work aims to provide a more detailed overview to allow an informed discussion about consensus protocol details. Hence, their work can be seen as supplemental literature, providing another angle on the topic.

Garay and Kiayias [52], in their SoK paper, provide a good overview over different consensus protocol properties and classify previous work according to those properties. It is especially well-suited to explore the design-space of consensus protocols.

XVII. Conclusion

While there exists a plethora of protocols, each exploring a slightly different direction, there is no clear winner in the race for the best protocol. Instead, there exist trade-offs in every solution, and the choice of a protocol mostly boils down to the properties one would like to emphasise.

This paper can serve as a basic overview of these trade-offs and directions to date. Moreover, it can be used as a collection of short descriptions and references to the protocols presented herein.

XVIII. Acknowledgements

We would like to acknowledge every member of the Nimiq team and also thank all the reviewers who contributed their time and their feedback to improve this paper, especially (in no particular order) Bruno França and Reto Trinkler.

REFERENCES

[1] F. Cristian, H. Aghili, R. Strong, and D. Dolev, *Atomic broadcast: From simple message diffusion to Byzantine agreement*. Citeseer, 1986.

[2] C. Cachin and M. Vukolić, "Blockchains consensus protocols in the wild," *arXiv preprint arXiv:1707.01873*, 2017.

[3] 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.

[4] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[5] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process." MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE, Tech. Rep., 1982.

[6] 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.

[7] 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.

[8] J. Gray and L. Lamport, "Consensus on transaction commit," *ACM Transactions on Database Systems (TODS)*, vol. 31, no. 1, pp. 133–160, 2006.

[9] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems (TOCS)*, vol. 16, no. 2, pp. 133–169, 1998.

[10] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm." in *USENIX Annual Technical Conference*, 2014, pp. 305–319.

[11] A. Zohar, "Securing and scaling cryptocurrencies," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, vol. 17, 2017, pp. 5161–5165.

[12] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *Communications of the ACM*, vol. 61, no. 7, pp. 95–102, 2018.

[13] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan, "Proofs of useful work." *IACR Cryptology ePrint Archive*, vol. 2017, p. 203, 2017.

[14] L. Luu, Y. Velner, J. Teutsch, and P. Saxena, "Smart pool: Practical decentralized pooled mining." *IACR Cryptology ePrint Archive*, vol. 2017, p. 19, 2017.

[15] A. Miller, A. Kosba, J. Katz, and E. Shi, "Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 680–691.

[16] 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.

[17] B. M. David, P. Gazi, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol." *IACR Cryptology ePrint Archive*, vol. 2017, p. 573, 2017.

[18] V. Buterin, "Slasher: A punitive proof-of-stake algorithm," *Ethereum Blog URL: https://blog. ethereum. org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm*, January 2014, [Online; posted 15-January-2014].

[19] 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.

[20] 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.

[21] I. Bentov, R. Pass, and E. Shi, "Snow white: Provably secure proofs of stake." *IACR Cryptology ePrint Archive*, vol. 2016, p. 919, 2016.

[22] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 583–598.

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

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

[25] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, 2016.

[26] V. Zamfir, "Introducing casper the friendly ghost," *Ethereum Blog URL: https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost*, August 2015, [Online; posted 01-August-2015].

[27] 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 *Security and Privacy (SP), 2016 IEEE Symposium on*. Ieee, 2016, pp. 526–545.

[28] Y. Sompolinsky and A. Zohar, "Accelerating bitcoins transaction processing," *Fast Money Grows on Trees, Not Chains*, 2013.

[29] Y. Lewenberg, Y. Sompolinsky, and A. Zohar, "Inclusive block chain protocols," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 528–547.

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

[31] C. LeMahieu, "Nano: A feeless distributed cryptocurrency network," *URL: https://nano.org/en/whitepaper*, 2018, [Online; accessed 10-December-2018].

[32] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *International Workshop on Public Key Cryptography*. Springer, 2003, pp. 31–46.

[33] A. Kiayias and A. Russell, "Ouroboros-bft: A simple byzantine fault tolerant consensus protocol," 2018.

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

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

[36] G. Danezis and S. Meiklejohn, "Centrally banked cryptocurrencies," *arXiv preprint arXiv:1505.06895*, 2015.

[37] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 17–30.

[38] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *Security and Privacy (SP), 2017 IEEE Symposium on*. Ieee, 2017, pp. 444–460.

[39] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel, "Measured performance of consistent checkpointing," in *Proceedings of the Eleventh Symposium on Reliable Distributed Systems*, no. LABOS-CONF-2005-052, 1992.

[40] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 439–453.

[41] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Symposium on Self-Stabilizing Systems*. Springer, 2015, pp. 3–18.

[42] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," *URL: https://lightning.network/lightning-network-paper.pdf*, 2016, [Online; posted 14-January-2016].

[43] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry, "Sprites: Payment channels that go faster than lightning," *CoRR abs/1702.05812*, 2017.

[44] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual International Cryptology Conference*. Springer, 1991, pp. 129–140.

[45] T. E. Jedusor, "Mimblewimble," 2016.

[46] A. Poelstra, "Mimblewimble," 2016.

[47] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for c: Verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 90–108.

[48] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 626–645.

[49] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," *Cryptol. ePrint Arch., Tech. Rep*, vol. 46, p. 2018, 2018.

[50] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *Bulletproofs: Short Proofs for Confidential Transactions and More*. IEEE, 2018, p. 0.

[51] 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.
[52] J. Garay and A. Kiayias, "Sok: A consensus taxonomy in the blockchain era," Cryptology ePrint Archive, Report 2018/754, 2018. https://eprint.iacr.org/2018/754, Tech. Rep.

APPENDIX

A. DLS

The paper "Consensus in the Presence of Partial Synchrony" [7] presents the so-called DLS algorithm. The authors thereof were the first to define models for achieving consensus in a partially asynchronous system. The proposed DLS algorithm works in a series of rounds divided into "trying" and "lock-release" phases:

In the first phase, each node communicates the value they believe is correct. The leader then *proposes* a value if enough nodes communicated this value. When a node receives a *propose* from the leader, it *locks* on the value and broadcast this information. If the leader receives enough *locks*, it *commits* to that value.

B. Cryptographic Primitives

1) Verifiable Random Functions: Using one's secret key sk, a VRF on any input string x returns two values: $VRF_{sk}(x) = (hash, \pi)$. The $hash$ is determined only by sk and x and is indistinguishable from random to anyone who does not know sk. The proof π allows anyone knowing the public key pk to verify that the hash indeed corresponds to the VRFs output for x.

2) Cryptographic Sortition: Cryptographic Sortition [20] relies on Verifiable Random Functions (VRF). As an input, Cryptographic Sortition takes a secret key sk, a seed $seed$, a role $role$, a threshold τ for the expected number of users selected for that role, the individual's weight w, as well as the total weight over all participants W.

It then uses a VRF to determine – proportionally to the weights – the amount of voting power the participant has with respect to the input. It returns a hash $hash$, a proof π, and the voting power j. Anyone can use these outputs and the public key pk to verify that the voting power j is indeed correct.

τ needs to be tuned such that enough, but not too many participants get elected.