SoK: Statelessness, State Management, and Scalability

Diego Espinosa, Silas Lenihan, Mariusz Derezinski-Choo

December 4, 2021

Abstract

Consensus protocols have traditionally required for participating nodes to store all of the states as the network progresses. Within these models, imperfect network conditions occur, with nodes dropping in and out of participation within the protocol. Without a stateless distributed system, nodes re-joining the protocol must depend on other nodes to gather information on the current state of the network. This leaves room for corruption depending on the intentions of the node that broadcasts the state of the network to the new node. Issues surrounding partially synchronous setting as described above has motivated statelessness within protocols. Statelessness allows nodes joining the network to learn about the current state from a fixed amount of data (e.g, the genesis block).

In this paper, we explore the current implementations of stateless protocols, and explore the road maps redefining older protocols that need to scale more efficiently with increased usage. Getting rid of many replicas of the same state history, paired with a simultaneous elimination of possible equivocation to newly participating node from the network creates promising protocol updates for future development. Given the limited storage in computers, investing in further exploration into statelessness has shown promise as distributed ledgers aim to take up a constant amount of space once statelessness is implemented onto a network. This issue only becomes more pertinent as the growth of large networks like Bitcoin and Ethereum.

Keywords. Stateless

Contents

1 Introduction

The current landscape within stateful blockchains his given rise to discussions surrounding statelessness, as both Bitcoin and Ethereum state sizes grow north of 200GB[18]. This explosion of state storage has created an entire new field of blockchain development where researchers aim to manipulate blockchains in order to decide how to reallocate state throughout participating nodes in order to make future blockchains lighter to store. An important delineation needs to be made as we delve further into the discussion of the need for statelessness on current blockchains. History is defined as past information that is kept for rebroadcasting or archiving purposes, but not critical for the process of the chain. History would include older blocks and receipts that may be kept by nodes. Although they are kept, the key point is that they are not state, since they aren't critical for the chain growth procedure. In contrast, state is the term for all information that nodes must store on a network in order to successfully participate in the chain growth procedure. For example, on the Ethereum network, nodes must store four different types of items in order to participate in the chain growth. First is account balances and nonces, which collectively agree on all of the accounts on the the network. Next, all smart contract code and required storage is kept in the state, which causes larger problems. Given that a lot of decentralized apps (dapps) are created on the blockchain everyday and given a one time use, these dapps abuse the storage of the network and create costs for every new node that needs to join the network and also store the state. Without proper storage etiquette, users on the network will abuse contract storage, leading to the explosion of storage for the Ethereum network that we see today. Lastly, consensus-related data is stored, but protocols in large part already optimize the amount of storage that this data takes up within a nodes state.

Without proper management of storage on the network, blockchains like Ethereum have a lot of junk state that has to be kept and downloaded by all nodes joining the network. The imbalance between the gas fee to add to the network as a miner and the perpetual cost to the network as a whole when state becomes larger creates a dangerous dynamic looking forward into the future of stateful blockchains. In practice, Ethereum contracts developers can use the self-destruct property in order to free up the storage used in the creation of the initial contract. The reality is that there is little incentive to partake in these best practices, hence the rapid growth of state on these networks.
This explosion of state creates a more expensive environment for new nodes to join the network. Maintaining an Ethereum node on AWS costs up to $70, and steadily increases with state given the amount that needs to be kept per node. This creates an unintended barriers to entry that is slowly decreasing the number of active nodes. With this decrease, the entire network becomes more centralized to the few nodes that are willing to take on the daily costs of hosting a node. This trend creates the exact opposite situation that blockchain network aim to foster, since decentralization is a vital part of their value proposition to the world.

As it stands, there are variations of statelessness that govern the ability of blockchains to provide nodes joining the network with adequate information about the state of the entire ledger.

Weak Statelessness Weak statelessness is characterized by the existence of a a function f, $f(I, S_t) = S_t$. where I is the initial state of the blockchain, and S_t is the set of all participating nodes and their local state at a given time t[12]. Weak statelessness is characterizes by having the proposing nodes have an entire state while all of the voting/validating nodes do not have to store a state at all. This important definition is the most popular topic when protocol developers talk about statelessness and its future in helping blockchains scale by requiring less storage as a network.

Partial statelessness Partial statelessness differs by having voting/validating blocks still carry some state with them, but only relative to some old state in the past. This means that any node on the netowrk that is active only needs to store a subsection of the entire history, which also decreases the amount of storage in the system as usage goes up. This version of stateless is the first step down from the current state-of-the-art that exacerbates the explosion of storage needed to maintain state.

The current technology behind many large blockchains requires a lot fo storage that becomes troublesome on systems with limited RAM. With the Unspent Transaction Set(UTXO) in bitcoin reaching

3GB, and 16GB for Ethereum [9], the "state explosion" has caused researchers problems as usage continues to rapidly expand. Without a fix, part of the network's state would have to be stored in secondary disks, leading to slower transaction validation, and higher vulberability to DoS attacks like the one that occured on Ethereum in 2016[9].

The rest of this paper is strucuted as follows, we first address multiple methods all centered around the effort of minimizing the amount of state that a active node has to carry, giving examples of current work that aims at improving scalability and/or reducing the potential for blockchain to have to take on different methods of storage outside of the local computer. We then explore some of the proposed designs for how statelessness can be implemented in order to immediately impact the performance of distributed ledger technologies. Lastly, we explore the Ethereum research around statlessness, as they lead the effort in implementing it into their ecosystem. We conclude with some final discussion of what lies ahead in the stateless space.

2 State Reduction

Prior to the introduction of statelessness, much work has been done to reduce the required amount of state stored on some or all nodes on the network. We classify theses approaches as *State Reduction*. Before discussing these strategies, it is prudent to give an overview of all the different state components that nodes must maintain. Legacy blockchains typically leverage three different data structures to carry out the protocol:

- **Transactions**, which consist of declarations that some amount of currency is to be transferred from one address/account to another
- **Blockchain Headers** which maintain the security of the blockchain via the inclusion of relavent nonce/hash fields that cryptographically prove that certain sets of transactions are to be includedon the block.
- **Account Balances**, which refer to the existing point-in-time balances of all the addresses/accounts in the protocol.

In Bitcoin, Node bootstrapping requires the retrieval of transactions and blockchain headers from other nodes on the network. Account balances are not retrieved because they are redundant in the sense that the ordered set of transactions included on the blockchain uniquely determines the current account balances (i.e. there is only one unique account balance ledger consistent with a particular set of transactions and block headers). This is because transactions essentially define the state transitions between each new iteration of the account balance ledger. For example, if at block i Steve has b_s Bitcoin and Berry has b_b Bitcoin, and block i includes a transaction of amount a from Steve to Berry, then the account balances for Steve and Berry in block $i+1$ must be $b_s - a$ and $b_b + a$ respectively as shown in figure 1. Balances are uniquely determined by transactions, but the converse is not true (clearly, a particular arrangement of account balances could be a result of infinitely many different transaction histories). Thus, many blockchains choose to bootstrap via transactions. However, bootstrapping via transactions is significantly more costly as the transaction history tends to be very large and grows at a relatively constant rate (over 360 GB as of October 21) whereas the account balance data structure is significantly smaller (3.5 GB in Bitcoin as of October 21) and grows at a significantly slower rate than the transaction history. For this reason, some state reduction strategies choose to bootstrap new nodes via account balances, allowing for significantly reduced memory and network requirements at the cost of the ability to query historical transaction data.

State Reduction strategies can be broadly categorized by two main approaches:

- **Trust Delegation** approaches, where nodes remove portions of their state in a way that makes them lose the ability to maintain the entirety of the original state. Trust delegating nodes instead rely on other nodes the complete state to answer queries related to the state that trust delegating nodes no longer have access to.
- **Blockchain Pruning**, where nodes internally optimize the structure of their state in a way that maintains the entirety of the original state. Nodes that undergo blockchain pruning still have

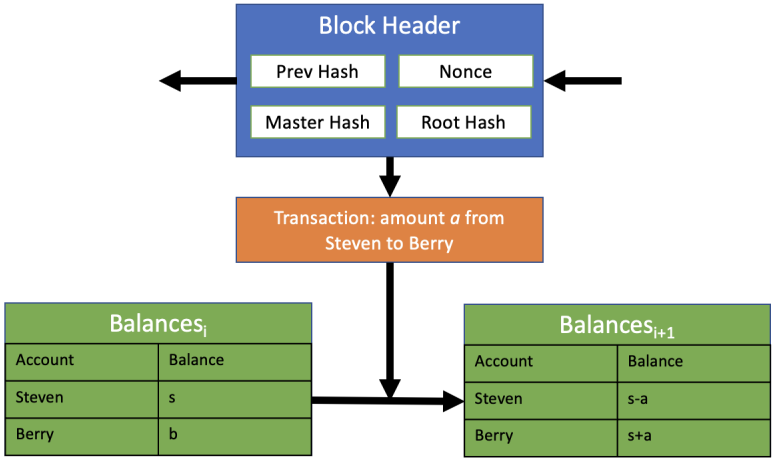

Figure 1: State Transition

access to the entirety of the state/ledger after pruning, allowing them to perform all relevant data lookups and transaction validations.

The dichotomy between trust delegation and blockchain pruning can best be conceptualized via its analogues to lossy vs lossless data compression. Trust delegation is similar to lossy data compression, where parts of the original state are irrecoverable by the node (except by querying a peer node on the network). Blockchain pruning is similar to lossless data compression, where the format of the state has been optimized, but the entirety of the original state is still reproducible by the node.

2.1 Trust Delegation

One simple and commonly adopted approach to the blockchain state problem involves some subset of nodes not executing the blockchain protocol in its entirety, but rather relying on some other trusted nodes (often referred to as "full" nodes) to perform some of the verification steps. This approach is inherently riddled with security concerns, as the network becomes centralized on the full nodes that perform the validation. The main idea of the blockchain is that its security should depend only on the cryptographic primitives underlying the chain itself, as opposed to some centralized authority(s), and trust delegation is the antithesis of this approach. Nonetheless, for completeness we summarize some of the widely adopted trust delegation approaches:

2.1.1 Light Nodes

Light Nodes [16] are nodes that opt to not download the entire blockchain, but rather download just the block headers for purposes of validating transactions. This allows the nodes to skip downloading the blockchain to perform verification, significantly decreasing the memory and network load on the system. However, lightweight nodes are extremely susceptible to fraudulent transactions as they only perform Simplified payment verification (SPV) to verify transactions, and do not validate the rules of the underlying application since they do not have access to all of the state information. For example, a light node may accept a transaction carried out with fake/invalid bitcoin if it requests transaction verification from a dishonest node. This attack vector can be mitigated by requesting transaction verification from several nodes. Nonetheless, no matter what precautionary measures are taken, there will always be a significant risk of fraudulent transactions because the lightweight node no longer relies on the cryptographic proofs underlying the blockchain, but rather relies on other nodes (which are very difficult/impossible to prove are honest).

2.1.2 Hot Wallets

Taking an even smaller responsibility than that of a light node, Hot Wallets [15] allow users to completely outsource fund management to a trusted third party who can issue transactions on their be-

half. This phenomenon has grown significantly as blockchains such as Bitcoin have started to achieve widespread adoption, as many people who own Bitcoin lack both the computing power and the technical expertise to host a node on the network. Services such as Coinbase and Binance perform transactions on the users' behalf. This approach is particularly susceptible to attack, as hot wallet services will have access to a large amount of the currency, creating a large incentive for attackers to try to breach the third parties and steal the currency.

2.2 Blockchain Pruning

Taking a more security-oriented approach than the Trust Delegation approaches, Blockchain pruning [10] aims to isolate components of the blockchain that nodes can safely delete, while preserving the ability to run the entirety of the protocol. In many cases, this requires restructuring the deleted data in a more optimized/compressed manner, so as to greatly reduce the memory footprint of the state that remains. The literature evaluates pruning algorithms based on three key metrics:

- **Scalability.** This includes the memory requirements needed for a full node on the network, the network bandwidth associated with bootstrapping new nodes, and the time/computational power required for a new node to be synchronized.
- **Security.** Safety of the blockchain is paramount, although the rigorous security requirements are deeply connected to the scalability concerns of existing blockchain technologies, as scalability and security tend to trade-off. Some pruning algorithms increase scalability at the expense of relaxed security assumptions
- **Backward Compatibility.** Many of the most well-established blockchains (in particular, Bitcoin) suffer from scalability problems the most. As a result, much of the literature is focused on finding solutions that can be added into existing technologies. In particular, Much emphasis is placed on solutions that can be *incrementally adopted*, or in other words, are compatible with some nodes running the new protocol and other running the old protocol. This is desirable as it prevents the network from having to perform a coordinated hard fork in order to adopt the new feature.

2.2.1 Simple Block Pruning

Simple block pruning [19] has been adopted into the Bitcoin client. The most straightforward of the approaches we will discuss, simple block pruning involves a new node bootstrapping the blockchain as per usual by downloading the longest chain from a peer and building the UTXO set by "replaying" all the transactions. Once the UTXO set is generated, the node can safely delete (i.e. *prune*) all of the old blocks, as the UTXO set is sufficient information to perform validation on incoming transactions.

From a scalability perspective, simple pruning only reduces the memory requirements of the nodes that utilize this approach, as they are able to discard of most of the transaction data of the blockchain. However, simple pruning does nothing to reduce the quantity of data needed to be transferred of the the network, as new nodes must still download the entire chain to recreate the UTXO set. In fact the network may be placed under additional stress since nodes that undergo simple pruning are unable to bootstrap new nodes, potentially forcing new nodes to fetch the chain information from further away nodes and over more costly channels. This makes them similar to light nodes the the sense that part of their functionality (bootstrapping new nodes) is lost. Additionally, the computational power required to synchronize a new node is completely unchanged, as the new node must still download the entire chain in order to compute the current local state.

2.2.2 Mini-Blockchain Pruning

The Mini-Blockchain Scheme is a pruning strategy based on synchronization of account balances, with strong similarities to other balance synchronization pruning protocols [2] [5] [7] The core observation underlying Mini-Blockchain's design is the fact that in Bitcoin, one mechanism is responsible for all of the core parts of the blockchain:

- Coordinating how the network processes transactions

- Securing the network via proof-of-work
- Managing account balances
- Maintaining historical records of the coins' ownership

In particular, the authors note that in Bitcoin synchronization, the historical transaction history and the account balances are tightly coupled via the contents of the blocks. This means that obtaining the current state requires "replaying" the entire history. While this makes sense from the perspective of having a single source of truth, it is the root cause of the growing cost of bootstrapping a new node on the network. To decouple these components of the blockchain, Mini-Blockchain splits the functionality across three distinct data structures:

The *account tree*, which is responsible solely for maintaining the balances of all non-empty addresses. When transactions are executed, the values of the relevant addresses of the account tree merely need to be recomputed, instead of adding new data and increasing the size of the data structure. This allows the account tree to be roughly constant in size over time (As new accounts/addresses get created, the tree will grow, although the size is still roughly bounded by the number of people would feasible own the currency, which provides an upper bound on its size. In particular, the account tree size does not grow with each new transaction). The account tree has a hash tree structure, with each node being assigned a hash of the hashes of its children. This builds up to a "master hash" at the root node of the tree, which is treated as a reference to uniquely identify the tree from the other parts of the chain.

The *proof chain*, which consists of the block headers. New nodes entering the network must download the entire proof chain to verify the chain with the highest cumulative proof of work. Since the hashes rely only on the block headers, nodes do not need to obtain old transactions to verify the proof of work on the chain. The proof chain maintains the proof of work to guarantee the security of the blockchain in the same way bitcoin does. in addition to including the master hash of the block, the proof chain headers include the root hash of the account tree. this cryptographically ties the account tree to the proof chain, allowing a node to verify that a particular account tree is associated with a certain block number. A visual representation of the proof chain is shown below

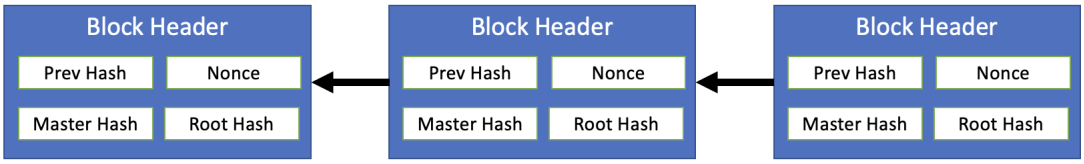

Figure 2: Structure of the Proof Chain

The *mini-blockchain* component, which is similar to the regular blockchain, with the key difference being that only the most recent transactions are kept. Nodes are able to prune old transactions because they are no longer needed to bootstrap a new node, since the balance information is encapsulated in the account tree.

2.2.3 Coin Prune

CoinPrune [10] is part of a family blockchain pruning strategies built on the concept of snapchat maintenance and synchronization [1] [6], as opposed to the balanced-based synchronization seen in minichain. Nodes periodically create snapshots of their local state (e.x. the current UTXO set in Bitcoin). When bootstrapping a new node, the snapshot is shared instead of the entire blockchain. This greatly reduces the storage requirements, network footprint, and synchronization time. Storage and network requirements are also reduced because the current UTXO set is significantly smaller than the entire blockchain.

To maintain the security of the synchronization process, snapchots must be publicly tied to the blockchain by the inclusion of a cryptographic identifier of the snapchot on the chain. In particular, the authors note that the coinbase transactions field of the bitcoin block may contain 100B of arbitrary data, which is currently ignored by nodes but may be used as a reference to the snapshots. This makes CoinPrune Bitcoin-compatible through a phase-in adoption process where it is possible for some nodes

to adopt CoinPrune but not others (since the non-adopted nodes will just ignore the references to the snapshots). Although the authors focus on Bitcoin backward compatibility, any blockchain with space in each block for some arbitrary data will be able to incrementally adopt CoinPrune.

To join the network, a node must first download a recent snapshot, and then download the headerchain, which consists of only the headers of each block on the blockchain. The node can use the headerchain to choose the blockchain branch with the most proof-of-work. From here, the node can retreive the chain tail, which is all of the block mined after the snapshot. The node can replay these blocks onto the snapshot to obtain the current state.

Since CoinPrune allows nodes to securely bootstrap via a snapshot, headerchain, and chaintail, nodes may save disk space by prune blocks prior to the snapshot. The main disadvantage of this approach is that historical data on previous transactions is not quereable, since the node now only has the current state and some of the recent transaction history. to remedy this, the authors suggest maintaining a small subset of archival nodes that maintain the entire blockchain in order to maintain the full historical transaction data.

3 Theoretical designs of Statelessness

3.1 EDRAX

EDRAX [4] is a cryptocurrency architecture which relies on short commitments contained in each consecutive block rather than relying on nodes storing the whole state of the blockchain. Because of the large state in most cryptocurrency systems (ie: Bitcoin, Ethereum), nodes are designed to be stateful, meaning that they keep some smaller state to keep track of the elements within the system. For Bitcoin (and others) "the validation state is a set of immutable coins called UTXO (unspent transaction outputs)" [4]. Other blockchains (Ethereum and others) maintain a set of mutable accounts in which a balance is maintained. In the UTXO model, a transaction is valid if it only uses unspent tokens, whereas in the account model a transaction is valid if the amount of tokens being transferred is less than the sending account's balance. These validation states are somewhat difficult to maintain and will continue to grow as these cryptocurrencies are more commonly used. EDRAX proposes a stateless validation architecture which is applicable to either the UTXO or account model.

In EDRAX, blocks contain a constant size commitment to the current validation state. Additionally, clients store a local proof that their coins can be spent with respect to the block commitment. When a transaction is made, it includes the digital signature from the sender as well as the local proof of their "account" status. After each commitment, users synchronize their local proofs.

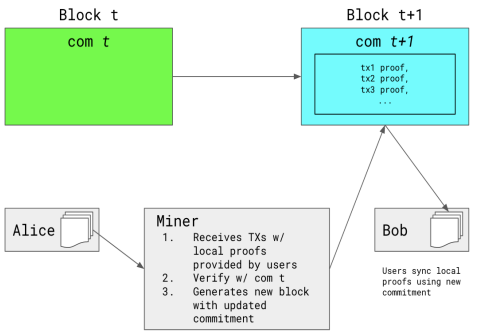

Figure 3: EDRAX Commitment Scheme

4 Practical implementations

4.1 Ethereum Stateless Roadmap

Ethereum is the leading blockchain for community built decentralized applications and cryptocurrency related projects. As mentioned previously, due to the widespread adoption of the Ethereum blockchain,

the state has reached and continues to grow to an increasingly problematic size, causing concerns of centralization as it becomes more difficult for an individual to operate a full node (see figure below). Thankfully, Ethereum's diverse developer community is rich with an ongoing discourse on ways to improve the system to achieve desired performance. There has been a large amount of ongoing research conducted by members of the Ethereum community relating to managing state size, therefore, we dedicate section 4.1 to outlining Ethereum's multi-staged implementation plan of state management. [3].

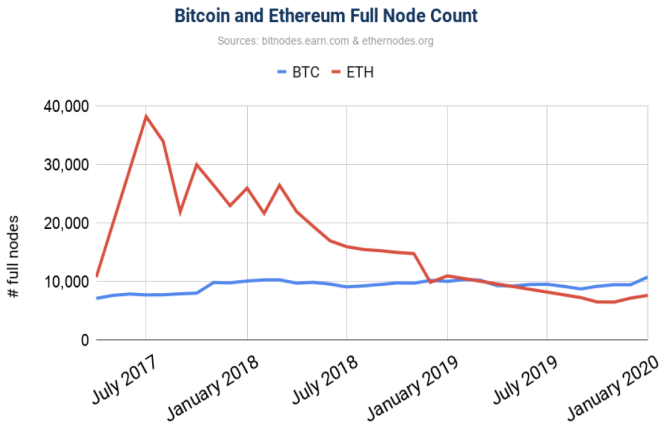

Figure 4: Number of BTC/ETH Full nodes over time [8]

The two main strategies to managing the state size of Ethereum are State expiry and weak statelessness. State expiry consists of removing a state that has not been recently accessed. This would reduce the state approximately to a flat 20-50 GB. Weak statelessness would require that only block proposers store the state, and allow block proposers to verify blocks statelessly. Vitalik Buterin and the Ethereum foundation have proposed a road-map to achieving both of these states in a well thought out way. These changes will likely come after the "merge" which is the first big step of Ethereum 2.0 [13].

4.1.1 Verkle Trees transition

The first state of Ethereum's stateless road-map is to introduce Verkle trees [11] as the way to store state. A hard fork would occur, where afterwards Verkle trees would store all edits to state and copy all of the accessed state, and the hexary Patricia tree would no longer be modifiable. Verkle trees allow for witnesses (proofs of data's existence in a state tree) to be reduced to a size of 200 KB per account. A Merkle Patricia tree uses hashes, whereas a Verkle tree uses vector commitments. A hash has a limitation by needing the entire vector to be passed to determine location and value. Instead, we can use vector commitments which contain an "opening" which allows us to verify a portion of the source data without revealing the entire vector. For a Merkle proof, you have to pass 15 siblings per level (7 on average), whereas for vector commitments you only need to pass the data at each level along with a commitment. This allows for a much smaller amount of data necessary to validate the source. Since we also don't need to reveal all siblings per level, each level can thus contain far more nodes, allowing for a shallower tree. They have chosen a tree structure with 256 and an estimated 4 levels of proof.

4.1.2 State Expiry

State Expiry [13] is the concept of having old state that isn't directly relevant being deleted from the state trees maintained by the nodes in the network. The core idea is that there is a single state tree per "period," approximately one year. After the end of a period, an empty state tree is initialized for the new period and any state updates will go to that tree.

Only the most recent tree can be modified, all older trees are immutable and their objects can only be made copies of that are included in newer trees and supersede the older copies. Full nodes only store the two most recent trees and don't need a witness for reading objects within those two trees, whereas reading older trees requires providing witnesses. A witness is a proof that verifies some value within the tree that one can verify only using the root of the tree. This proof needs to show that the data was present in the period 2 periods ago. If you are trying to resurrect data in n periods ago, for all $n > i >= 2$ you must submit a proof of absence at all periods i. This, of course, results in a higher cost for resurrection. Addresses will also be extended to include the period that the transaction takes place in.

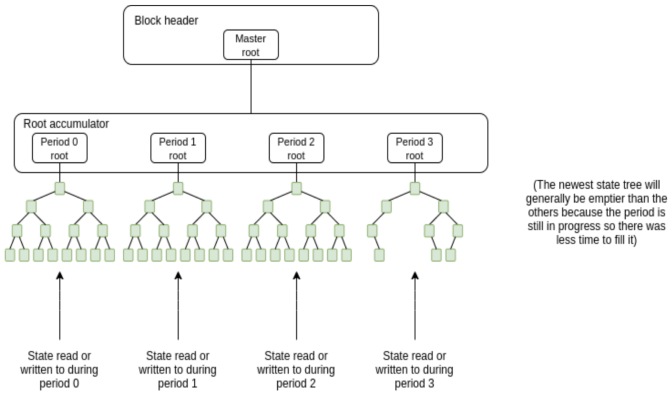

Figure 5: Diagram of State Expiry [14]

4.1.3 State Networks and Portal network

State networks are another useful tool for minimizing the state that full nodes are required to store. The idea is to provide a parallel network to the block propagation network in which each node only stores a fraction of the data which holistically represents the entire state. Users can query this network on an as-needed basis. Therefore, validators don't need to store any data relating to the state whatsoever. The data is validated via a Verkle or Merkle proof, however, this detail is still under debate.

4.1.4 State transition

State transition is a key idea for weak statelessness, as it defines the way in which the minimized state is updated as more blocks are added to the chain. Suppose we can define a state transition by a function $STF(S, B) \rightarrow S$'. S and S' are states and B is a block (or transaction T). The root state of S is a 32 byte root hash of a Merkle tree containing S. B contains a witness W which is a set of Merkle branches proving the values of all data that the execution of B accesses. The STF function takes a state root, block, and witness as an input and outputs the new state based on the information provided by the witness. Full nodes therefore only would store state roots and miner's would be responsible for packaging Merkle branches (witnesses) along with blocks which full nodes would verify and download.

4.1.5 Statelessness and Sharding

Unlike in Ethereum 1.0, Ethereum 2.0 will implement stateless miners by leveraging the techniques described above. Such an upgrade will be necessary for implementing sharding[21], a key upgrade to the scalability set to be released as a major part of Ethereum 2.0. Sharding is the concept of spreading the workload of the network across segmented groups to greatly increase transaction throughput. Due to this massive upgrade to transaction throughput, stateless miners will be required to keep up with the network to validate their given shard. Additionally, Sharding enables verifiers to only store data for their shard for a given epoch (a period containing 32 slots, which each contain a new block), so some state management technique must be put in place to ensure the validity of a given shards

information with respect to the rest of the network. Once again, due to the nature of ongoing research on Ethereum, there isn't yet an exact agreed upon solution to this problem.

4.2 Mina Protocol

Mina[8] is a novel blockchain system that aims to create a Succinct Blockchain, in which every block can be verified in constant time by leveraging the power of Zero Knowledge Proofs. By using recursive ZK-Snarks, the Mina Protocol is able to create block commitments that attest to the validity of the entire state, and are able to be stored in a constant 22 Kilobytes, a minuscule figure. In other words, each block commitment is a ZK-Snark proving the current blockchain state, while also including a compressed version of the previous blockchain states. This allows for the proof maintained by nodes to remain incredibly small. Such an accomplishment is a massive breakthrough for blockchain systems, as it allows anyone with a smartphone to be capable of transactions verification. In addition to the provably secure Proof of Stake Ouroboros Samasika consensus protocol, Mina provides a revolutionary cryptocurrency model.

In practice, Mina is able to create reliable, scalable, decentalized applications called "Snapps," which are applications that handle verifications off-chain, while preserving the security and privacy of traditional blockchains. Although Snapps are largely still in the development stage, they promise efficient, trustless execution akin to smart contracts, while avoiding many of the pitfalls of popular smart contract blockchains such as Blockchain. While Mina promises a lot, and its ZK-Snark infrastructure has proven to be effective at maintaining blockchain state, there is one real point of concern with their architecture.[17] According to their documentation, "for some usecases, it is useful to maintain this historical data," [20]. Many critics are skeptical of whether these archival nodes are truly decentralized, and believe they may present a single point of failure for many key Mina processes. According to their documentation, Mina archive nodes can store the archived data in a local postgres database, however, for redundancy, most nodes use specifically Google Cloud for storage. This has been identified as a single point of failure for Mina's system, and it goes against the general idea of decentralization that is the basis for blockchains in the first place. Although it is still unclear exactly how reliant Mina is on these archive nodes, it is still an ongoing point of concern for those who analyze the technology behind Mina. Nonetheless, Mina is a great example of a stateless design for a powerful blockchain and cryptocurrency system, and once they work out the kinks of their design, it could be a very important blockchain of the future.

5 State Management Model Comparison

In this section, we provide a summary of the main state management models, listing their advantages and disadvantages. The techniques within each protocol solves different aspects of the state management problem while simultaneously raising their own unique challenges. We provide an overview of the individual promises of each protocol, as well as the problems associated with each.

Table 1: Summary of advantages and disadvantages of different state management models

Model	Advantages	Disadvantages
Trust Delegation	• Extremely small memory footprint as nodes only need to store block headers • Allows for significantly more nodes to enter the network (although these additional nodes do not perform the entire protocol)	• Significant security risks • Creates network centralization as only a subset of "full nodes" perform all of the necessary validation on transactions
Simple Block Pruning	• Backwards compatible with existing protocols (in particular, Bitcoin) • Nodes can discard transaction history (360GB of state)	• Nodes lose the ability to bootstrap new nodes on the network • Ability to query historical transaction data is lost • Does not reduce network load as synchronization still requires full download of entire transaction history
CoinPrune and Mini-Blockchain	• Nodes can discard transaction history (360GB of state) • Significant reduction in network load as synchronization depends on account balances (4GB for bitcoin) instead of transactions (360GB for Bitcoin)	• Ability to query historical transaction data is lost on nodes performing pruning • Transaction data becomes available only on a limited subset of nodes, leading to network centralization with respect to parties attempting to query historical transaction information
EXRAX	• Constant size, lightweight proofs to verify blockchain state. • No nodes must store the entirety of the blockchain	• Incompatibility with Merkle Proofs, which would result in a serious overhaul of many blockchain systems. • Slow speed of vector commitment proof model. • Cost to users associated with storing their own proofs.
Ethereum Stateless Design	• Comprehensive step-by-step implementation plan with detailed chronology. • Gradually shifts to more manageable state and better performance beginning with Verkle Trees transition and State expiry. • State expiry results in flat 20-50 GB stores per full node, a massive improvement. • Enables efficient sharding, allowing for state improvements and throughput scalability.	• No completely stateless solution. • Highly conceptual at the moment, there isn't an agreed upon solution in development yet. • Due to the Ethereum community's focus on implementing Proof of Stake and other improvements first, it may be a long time before stateless solutions are developed.
Mina Protocol	• Extremely small (22 KB), constant state size, due to ZK-Snarks. • Snapps provide lightweight, secure, off-chain application execution. • Snapps allow for trust-less web-oracles, high privacy applications, and efficient decentralized identity implementations. [17] • Increased decentralization due to the ability to run a full node on basically any modern device, including smart phones.	• Due to the time required to created a ZK-Snark, transaction throughput is quite limited. Additionally, it takes one hour for a transactions to become finalized, similar to Bitcoin. Such slow transactions throughput and finalization is not ideal for a broadly used payment network. • The single point of failure associated with Archival nodes. • Many features of Mina are still in development and somewhat theoretical.

6 Conclusion and open challenges

Statelessness, although novel in its implementation inside of the blockchain space, has been applied in the realm of server requests for some time. Statelessness inherently brings with it many advances, but alongside its implementation comes an overhaul of stateful protocols that require their native blockchains to store information on its entire state. The implementation of statelessness can compromise the protocols, but as many see it, it is worth the work to bring about true scalability in future blockchains. Nodes no longer being required to store state brings about a new question, who does store the state now? Many popular blockchains had a smooth user experience, since users did not have to worry about storing private state. The question becomes how statelessness can continue to provide the same smooth experience once it is implemented in widely-used blockchains like Etheruem.

Additionally, many blockchains aim to move towards a Proof-of-Stake(PoS) concept in order to improve the energy usage and security aspects behind the traditionally implemented Proof-of-Work protocols. In its current state, statelessness and PoS protocols being implemented together brings a new host of challenges as blockchain developers and researchers aim to optimize issues surrounding energy usage, scalability, speed, and security.

In this Systematization of Knowledge, we broke down the relevant research being done in the field of statelessness, defined different forms of statelessness, and presented both theoretical and practical implementations of various state management protocols. We defined the concepts of weak, strong, and probabilistic statelessness, and discussed the advantages of each. We outlined Light Nodes, Hot Wallets, and discussed various forms of blockchain pruning: the concept of discarding unnecessary information once a valid blockchain state has been confirmed. Further, we discussed theoretical implementations of statelessness via EDRAX, a protocol that uses short commitments store within each block to verify the state. Additionally, we discussed noteworthy practical implementations of statelessness, including a detailed breakdown of the Ethereum Stateless Road-map and a summarization of the research efforts being done in the Ethereum community. This multistage road-map consists of implementing state expiry, verkle trees, and eventually statelessness. Further, we discussed the Mina Protocol, a promising new Blockchain solution that utilizes revolutionary ZK-Snarks to create recursive zero knowledge commitments of the blockchain state, allowing for a lightweight and constant size state to be maintained by full nodes. Lastly, we compared the advantages and disadvantages of the different stateless models discussed in the paper, providing a general outlook of the current state of the art.

In summary, state size is a major limitation of modern blockchains and threatens the very decentralization that makes blockchain systems desirable. There is an ongoing research effort across the community to create viable implementations of statelessness in order to solve this scalability problem. The goal of this SoK was to define different forms of statelessness, provide a comprehensive view into the current state of research, and demonstrate the direction of practical implementations.

References

[1] Alexander Chepurnoy, Mario Larangeira, and Alexander Ojiganov. *Rollerchain, a Blockchain With Safely Pruneable Full Blocks*. 2016. arXiv: 1603.07926 [cs.CR].

[2] Tom Elvis Jedusor. "MIMBLEWIMBLE". In: (2016). URL: https://docs.beam.mw/Mimblewimble.pdf.

[3] Vitalik Buterin et al. *The stateless client concept*. Oct. 2017. URL: https://ethresear.ch/t/the-stateless-client-concept/172.

[4] Alexander Chepurnoy et al. "EDRAX: A Cryptocurrency with Stateless Transaction Validation". In: (2018). URL: https://eprint.iacr.org/2018/968.pdf.

[5] Derek Leung et al. "Vault: Fast Bootstrapping for the Algorand Cryptocurrency". In: *Network and Distributed Systems Security (NDSS) Symposium 2019* (2019). URL: https://www.mit.edu/~yossigi/vault.pdf.

[6] Alexander Marsalek et al. "Tackling Data Inefficiency: Compressing the Bitcoin Blockchain". In: *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2019, pp. 626–633. DOI: 10.1109/TrustCom/BigDataSE.2019.00089.

[7] Herman Schoenfeld. "Pascal: An Infinitely Scalable Cryptocurrency". In: (2019). URL: https://www.pascalcoin.org/storage/whitepapers/PascalWhitePaperV5.pdf.

[8] Joseph Bonneau et al. *Mina: Decentralized cryptocurrency at Scale*. Mar. 2020. URL: https://minaprotocol.com/wp-content/uploads/technicalWhitepaper.pdf.

[9] Huan Chen and Yijie Wang. "MiniChain: A lightweight protocol to combat the UTXO growth in public blockchain". In: *Journal of Parallel and Distributed Computing* 143 (2020), pp. 67–76. DOI: 10.1016/j.jpdc.2020.05.001.

[10] Roman Matzutt et al. "How to Securely Prune Bitcoin's Blockchain". In: (2020). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9142720.

[11] Guillaume Ballet. *Statelessness amp; Verkle Trees*. Oct. 2021. URL: https://www.youtube.com/watch?v=f7bEtX3Z57o&ab_channel=ETHGlobal.

[12] François Bonnet, Quentin Bramas, and Xavier Défago. "Stateless Distributed Ledgers". In: *Networked Systems Lecture Notes in Computer Science* (2021), pp. 349–354. DOI: 10.1007/978-3-030-67087-0_22.

[13] Vitalik Buterin. *A State expiry and statelessness roadmap*. 2021. URL: https://notes.ethereum.org/@vbuterin/verkle_and_state_expiry_proposal.

[14] Vitalik Buterin. *A State expiry and statelessness roadmap*. 2021. URL: https://notes.ethereum.org/@vbuterin/verkle_and_state_expiry_proposal.

[15] *Hot Wallet*. 2021. URL: https://en.bitcoin.it/wiki/Hot_wallet.

[16] *Lightweight node*. 2021. URL: https://en.bitcoin.it/wiki/Lightweight_node.

[17] Alpha Trades. *A snarky examination of Mina Protocol-the world's "lightest" Blockchain*. June 2021. URL: https://alphatrades.medium.com/a-snarky-examination-of-mina-protocol-the-worlds-lightest-blockchain-b9826f6799de#e354.

[18] *A Theory of Ethereum State Size Management*. URL: https://hackmd.io/@vbuterin/state_size_management.

[19] *Bitcoin Core version 0.11.0 released*. URL: https://bitcoin.org/en/release/v0.11.0#how-to-upgrade.

[20] *Mina Docs - Archive node*. URL: https://docs.minaprotocol.com/en/advanced/archive-node.

[21] *Stateless clients*. URL: https://docs.ethhub.io/ethereum-roadmap/ethereum-2.0/stateless-clients/.