How to Securely Prune Bitcoin's Blockchain

Roman Matzutt*, Benedikt Kalde*, Jan Pennekamp*, Arthur Drichel†, Martin Henze‡, Klaus Wehrle*
*Communication and Distributed Systems, RWTH Aachen University, Germany · {lastname}@comsys.rwth-aachen.de
†IT Security Research Group, RWTH Aachen University, Germany · drichel@itsec.rwth-aachen.de
‡Cyber Analysis & Defense, Fraunhofer FKIE, Wachtberg, Germany · martin.henze@fkie.fraunhofer.de

***Abstract*—Bitcoin was the first successful decentralized cryptocurrency and remains the most popular of its kind to this day. Despite the benefits of its blockchain, Bitcoin still faces serious scalability issues, most importantly its ever-increasing blockchain size. While alternative designs introduced schemes to periodically create snapshots and thereafter prune older blocks, already-deployed systems such as Bitcoin are often considered incapable of adopting corresponding approaches. In this work, we revise this popular belief and present *CoinPrune*, a snapshot-based pruning scheme that is fully compatible with Bitcoin. CoinPrune can be deployed through an opt-in velvet fork, i.e., without impeding the established Bitcoin network. By requiring miners to publicly *announce* and jointly *reaffirm* recent snapshots on the blockchain, CoinPrune establishes trust into the snapshots' correctness even in the presence of powerful adversaries. Our evaluation shows that CoinPrune reduces the storage requirements of Bitcoin already by two orders of magnitude today, with further relative savings as the blockchain grows. In our experiments, nodes only have to fetch and process 5 GiB instead of 230 GiB of data when joining the network, reducing the synchronization time on powerful devices from currently 5 h to 46 min, with even more savings for less powerful devices.**

***Index Terms*—blockchain; block pruning; synchronization; bootstrapping; scalability; velvet fork; Bitcoin**

I. Introduction

Bitcoin [1] and its (public) blockchain, an immutable append-only ledger of financial transactions, constitute an ongoing success story. Via the blockchain, all nodes can independently verify Bitcoin's history and thus mutually distrusting peers can establish consensus about the correctness of those transactions, which now also fuels applications such as audit systems [2]–[4], transparency overlays [5], [6], anonymity bootstrapping services [7], and smart contracts [8].

However, Bitcoin also serves as a prime example of the scalability challenges of widely used blockchain systems. Among these challenges are, e.g., limited transaction throughput, high payment verification delays [9], and, most importantly, ever-growing blockchain sizes. For instance, Bitcoin's blockchain has a size of 253 GiB with a recent average growth rate of 139 MiB/d as of Apr 13, 2020 [10]. These high demands cause individual nodes to *prune* historical blockchain data [11], i.e., older payment flows that have been superseded by newer ones. While such decisions are rational from the user's perspective, they harm Bitcoin's overall network health: To root trust into Bitcoin's current state, new nodes need to obtain and reverify *all* blockchain data, including all historical data. Given the decentralization of public blockchains, which are designed to avoid trusted entities, new nodes require independent sources to obtain all bootstrapping information.

While joining nodes would benefit most if every single node held a full copy of the blockchain, already synchronized nodes need to save disk space instead. To resolve this inherent conflict of interests, alternative designs [12]–[15] proposed *snapshot-based synchronization*, where new nodes do not verify all historical data but rely on a recent snapshot of the blockchain's state. However, well-established blockchain systems, such as Bitcoin, have especially high demands to implement performance improvements but are hard to adapt at the same time: Major changes have to be adopted by a majority of nodes to prevent permanent network partitioning, which has proven difficult in the past [16]. Based on these observations, we identify and raise two main questions: ***(a)*** *how to extend existing blockchain systems with pruning capabilities and* ***(b)*** *how to do so in a secure and trustworthy manner?*

To answer both questions, we first survey approaches to reducing blockchain sizes, arguing that they either are inefficient, insecure, or not deployable to existing systems. We thus propose *CoinPrune*[1], a block-pruning scheme that is fully compatible with Bitcoin and can be adopted immediately by any subset of nodes. CoinPrune enables joining nodes to synchronize via recent and *trustworthy* snapshots of Bitcoin's state. By establishing additional trust into these snapshots in a distributed manner, CoinPrune drastically unburdens all Bitcoin nodes: First, new nodes only need to obtain a small fraction of the blockchain. Further, CoinPrune enables all nodes to prune obsolete information *without* affecting the overall network health. CoinPrune establishes trust by periodically having miners cryptographically tie snapshots to the blockchain while other miners *independently reaffirm* the snapshots' correctness. Thus, assuming an appropriate partial adoption of our scheme, joining nodes may rely on the honest network majority to verify snapshots. Reaffirmations are ignored by legacy nodes and do not affect block acceptance. Hence, CoinPrune can be deployed mid-operation via a velvet fork [17], [18]. Consequently, new nodes only require a recent reaffirmed snapshot and subsequent full blocks to synchronize.

Our evaluation shows that full nodes and miners supporting CoinPrune can reduce disk space utilization by 86 %. Still, they are able to help joining nodes synchronize. In fact, CoinPrune drastically improves synchronization performance: While network traffic generated by joining nodes is reduced by 93 %, synchronization time drops from 5 h to 46 min on powerful devices and even more for less powerful devices.

[1]Research prototype available at https://github.com/COMSYS/coinprune

© IFIP, 2020. This is the author's version of the work. It is posted here by permission of IFIP for your personal use. Not for redistribution. The definitive version was published in *IFIP NETWORKING 2020*.

II. Bitcoin Overview

We start off with a primer on Bitcoin before describing its transaction management, blockchain layout with implications on consensus updates, and its bootstrapping process.

Bitcoin Primer. Bitcoin's [1] main contribution was its blockchain, a public and immutable append-only ledger of financial transactions to prevent the double-spending of coins within an untrusted peer-to-peer (P2P) network. Bitcoin establishes this ledger by bundling pending transactions in cryptographically interlinked, hard-to-create blocks. The blockchain is jointly maintained by a P2P network of *full nodes* that locally verify pending transactions and blocks, discarding any incorrect information. Special nodes, the *miners*, invest their computational power to create new blocks by solving a proof-of-work (PoW) puzzle in exchange for freshly minted bitcoins as a reward. Modifying blocks at a later point becomes increasingly hard as it requires recomputing all subsequent blocks to keep their chaining intact.

Transaction Management. Each transaction transfers previously received or minted bitcoins to one or more receivers via individual *transaction outputs*. To prevent double-spending, full nodes have to verify claimed coin ownership for all pending transactions. For efficiency reasons, all nodes keep track of the current *set of unspent transaction outputs (UTXO set)*. Thereby, nodes can discard pending transactions that attempt to spend non-existing or already spent bitcoins. Notably, by default, full nodes prune all spent transactions from their transaction index, i.e., if they do not contribute to the UTXO set anymore. Nevertheless, these nodes retain a full copy of historical blocks to help bootstrap new nodes.

Blockchain Layout and Consensus Updates. Each Bitcoin block consists of a header and a set of transactions attached to the block via a Merkle tree. The 80 B-long header consists of a version field, the hash value of the block's predecessor for chaining the blocks, the Merkle tree root to cryptographically tie the transactions to the block header, an approximate timestamp, as well as the miner's PoW. Given the distributed nature of Bitcoin, blockchain *forks*, i.e., situations in which the blockchain diverges into more than one potential path forward, can occur either accidentally or with intent. Accidental forks occur when multiple miners find valid blocks concurrently. These forks are resolved naturally since one branch is highly likely to grow faster (i.e., accumulate more PoW), causing all nodes to abandon other branches in favor of the fastest-growing branch. Intentional forks are used to *update* existing consensus rules and are traditionally categorized as either *hard forks* or *soft forks* [18]. While hard forks introduce protocol-breaking changes to the consensus rules, e.g., altered block structures, soft forks aim to remain backward-compatible with clients following older consensus rules [18]. Both paradigms can incur *permanent* blockchain forks depending on whether the majority of nodes accepts or rejects the proposed changes. Contrarily, multiple works [17], [18] recently investigated *velvet forks*, which aim to allow for the *gradual introduction* of new features without creating permanent forks. This type of fork augments upgraded blocks in a way that is still valid to legacy nodes, while updated nodes process them in accordance with the changed protocol.

Initial Synchronization. When a node first joins the Bitcoin network, it needs to obtain its individual view on Bitcoin's current state of consensus, i.e., the UTXO set resulting from the blockchain path containing most PoW. To keep this process fully decentralized and independent from trusted nodes, each node initially establishes eight outgoing connections to random established nodes, called *neighbors*, and downloads the complete blockchain from them. Due to the separation of headers and transactions, nodes first fetch the *headerchain*, i.e., chain of block headers, and simultaneously request full blocks, i.e., the corresponding transactions. While receiving the data, the joining node verifies its correctness by (a) verifying the blockchain's cryptographic links back to the hard-coded genesis block, (b) keeping track of the amount of performed PoW to remain on the currently valid blockchain, (c) validating transaction sets tied to each block, and (d) by checking the correctness of transactions and replaying them to obtain an up-to-date UTXO set. Even though this information is sufficient to process newly mined blocks, nodes keep a full copy of the blockchain by default.

III. Impact of Growing Blockchain Sizes

By design, blockchains continuously grow in size and thus eventually reach prohibitive sizes. For instance, the most popular system, Bitcoin, suffers from a blockchain size of currently 253 GiB with a recent average growth rate of 139 MiB/d as of Apr 13, 2020 [10]. This worrying trend severely impacts the scalability of the overall network. In addition to increasing disk space requirements, which already today exclude devices such as smartphones from running a full Bitcoin node, we observe negative influences of bandwidth requirements, processing costs, and synchronization times of newly joining nodes.

Storage Requirements. To retain a decentralized consensus network, Bitcoin requires that enough independent nodes persistently maintain a full blockchain copy to help bootstrap joining nodes (cf. Section II). However, storing hundreds of Gigabytes of historical blockchain data is both irrational for individual node operators and prohibitive on storage-constrained devices. In consequence, such devices cannot act as full nodes, and users have to accept weakened security guarantees.

Bandwidth Requirements. During initial synchronization, each joining node must obtain a full blockchain copy. Current blockchain sizes already require good Internet connectivity for both the joining node and its serving nodes, potentially causing increased initial synchronization times for joining nodes. Furthermore, such requirements also put an additional burden onto existing nodes as serving new nodes consumes resources that could otherwise be used for other tasks, e.g., gossiping pending transactions or newly mined blocks.

Processing Costs. In addition to downloading the blockchain, joining nodes also need to verify the blockchain's integrity and locally replay every single transaction to build the UTXO set. This process consumes excessive amounts of computation power for joining nodes. Especially, the presence of large numbers of obsolete transactions that do not contribute to the UTXO set wastes valuable resources anymore.

Synchronization Time. The combination of high bandwidth requirements and high processing costs cause prolonged synchronization times. While benchmarks using powerful clients report about 5 h in 2018 [19], literature already highlighted this issue in 2016 when four days were required to synchronize Amazon EC2 nodes [9]. Naturally, this problem aggravates over time as new blocks are added continuously.

In summary, the ever-growing blockchain heavily impedes the scalability of the overall system for both joining and existing nodes. Existing nodes are even punished for acting altruistically in the network by helping joining nodes synchronize. These problems are especially severe for popular systems such as Bitcoin. In the following, we survey to which extent (newly proposed) schemes attempt to tackle these issues.

IV. The Current State of Blockchain Pruning

Blockchains, especially public ones, have long suffered from their limited scalability. Consequentially, developers have tackled these scalability challenges from different perspectives. In this section, we survey current state-of-the-art measures deployed to existing systems as well as alternative blockchain designs that focus on reducing storage requirements and improving the bootstrapping. Other works that consider blockchain data management, but are not covered here explicitly, include analyses of blockchain data [32]–[36] and the UTXO set [37], approaches to prevent illicit content from being engraved into the blockchain [38]–[43], sharding approaches [44]–[46], and lightweight payment schemes [47], [48].

A. Survey Criteria and Methodology

We qualitatively assess the applicability and effectiveness of related approaches based on their (a) scalability improvements, (b) whether they maintain sufficient levels of security, (c) their impact on the overall network, (d) potential impact on blockchain queryability, (e) and their compatibility with already established public blockchain systems such as Bitcoin.

Regarding *scalability improvements*, we separately consider processing, traffic, and storage. Based on these improvements, we discuss the overall impact on synchronization times for joining nodes. We resort to a qualitative assessment of the presented approaches, as most works do not present performance benchmarks. We comment on the approaches' *security* by discussing whether or not these proposals actively weaken security guarantees of the base system. Furthermore, we discuss the *impact on the overall network* by considering the network health, i.e., the dependency on especially altruistic nodes, and the potential overhead the approaches may introduce for already synchronized nodes. Then, we assess how the proposed schemes may impact the *blockchain queryability*, e.g., the capability of querying historical transactions or augmented transactions such as Bitcoin's `OP_RETURN` transactions. Finally, we survey their *compatibility* with already deployed blockchain systems. We summarize our results in Table I.

B. Measures Deployed in Existing Blockchain Systems

The increased popularity of cryptocurrencies forced their developers to tackle rising scalability issues. In this section, we present measures taken either by users locally or by blockchain developers network-wide. Our discussion is based on the reference implementations (Bitcoin Core and Ethereum's `geth`, respectively) where appropriate. Overall, we identify approaches based on *trust delegation*, *skipping verification* steps, *improving data management* with the special case of *block pruning*, and *state-based synchronization*.

Trust Delegation. Users can delegate their trust into the blockchain's correctness to third parties if they cannot operate a full node, e.g., when using a constrained device for issuing transactions. Using *hot wallets* [20], users essentially outsource all fund management to a trusted third party, enabling the service provider to issue transactions on their behalf. Similarly, *light nodes* [21] outsource blockchain verification to other full nodes, but they manage their wallet locally using simplified payment verification (SPV) [1]. These approaches vastly improve the performance of clients, only put a negligible burden on the full nodes, and they are actively used. However, they only seize the resources of other nodes and do not contribute positively to the overall network. Contrarily, trust-delegating nodes heavily rely on a backbone network of full nodes for both trust and relevant information and prohibit local verifiability. The never deployed Ultimate Compression scheme [22] aimed at bootstrapping light nodes with the current UTXO set but requires full nodes to store and transmit a searchable representation of the UTXO set in addition to its full blockchain copy, putting extra burden on the full nodes. Furthermore, this scheme requires an additional blockchain to establish trust in the transmitted UTXO set.

Improving Data Management. Increasing blockchain sizes necessitate optimized data management, either for looking up relevant information or for efficiently bootstrapping new nodes. To this end, Bitcoin Core has historically changed its *underlying database* system [23] and the internal layout of its UTXO set [24]. Furthermore, full nodes *locally prune* irrelevant entries from their *transaction index* [23]. While the raw blockchain data is still persisted, historical information is not queryable anymore. Network-related optimizations mainly engulf a revised *header-first download* of the blockchain [25]. Verifying the headerchain is sufficient to ensure the blockchain's integrity. Since transactions can be decoupled from their block's headers (cf. Section II), nodes can now download and verify full blocks in parallel with only minor and local upgrading incompatibilities [25]. Nodes can further *limit their block-serving bandwidth* [49], and they can relay *compact representations* of newly mined blocks, thereby avoiding transmitting known-but-pending transactions redundantly [50]. However, header-first download still requires to transfer and process all blockchain data during initial synchronization, only its distribution is more efficient.

Skipping Verification. Bitcoin's reference implementation started early on to avoid reverifying transactions from very

TABLE I
QUALITATIVE COMPARISON OF APPROACHES IMPROVING STORAGE REQUIREMENTS AND INITIAL SYNCHRONIZATION

	Name	Approach	Reduce Processing	Reduce Traffic	Reduce Storage	Sync. Time	Maintain Security	Network Health	Server Burden	Comple-teness	Compat-ibility
Deployed Solutions	Hot Wallets [20]	Trust Delegation	○ / ●	○ / ●	○ / ●	○ / ●	○	○	●	○	●
	Light Nodes [21]	Trust Delegation	○ / ●	○ / ●	○ / ●	○ / ●	◐	○	●	○	●
	"Ultimate Compression" [22]	Trust Delegation	○ / ●	○ / ●	○ / ●	○ / ●	○	●	○	●	●
	DB Improvements [23], [24]	Data Management	◐	○	○	◐	●	●	●	●	●
	Index Pruning [23]	Data Management	◐	○	◐	◐	●	●	●	◐	●
	Headers-first Download [25]	Data Management	○	○	○	◐	●	●	●	●	◐
	Assume-valid Blocks [26], [27]	Skip Verification	●	○	○	◐	●	●	●	●	●
	Block Pruning [11]	Simple Block Pruning	○	○	●	○	●	○	●	◐	●
	Ethereum Fast Sync [28]	State-based Sync.	●	○	○	●	●	●	●	●	○
Related Work	Selective Pruning [29]	Simple Block pruning	◐	◐	●	◐	○	●	○	●	○
	Rollerchain [14]	State-based Sync.	●	●	●	●	●	●	◐	◐	○
	Marsalek et al. [30]	State-based Sync.	○ / ●	○ / ●	○ / ●	●	●	◐	◐	◐	○
	Mini Blockchain Scheme [12]	Balance-based Sync.	●	●	●	●	●	◐	◐	○	○
	Mimblewimble [13]	Balance-based Sync.	○	◐	◐	○	●	●	●	○	○
	Pascal [15]	Balance-based Sync.	●	●	●	●	◐	○	●	◐	○
	Vault [31]	Balance-based Sync.	●	●	●	●	●	●	●	◐	○
	CoinPrune (our approach)	State-based Sync.	●	●	●	●	●*	●*	◐	◐	●

□ / □: *Distinction Full Nodes / Light Nodes* ∗: *Dependent on honest majority among adopters (cf. Section VIII)*

old blocks. Using hard-coded *checkpoint blocks* at first [26], Bitcoin has recently shifted to use configurable *assumed-valid blocks* [27]. The reasoning here is that invalid transactions would have been rejected by the network earlier, and thus older transactions with many confirmations are believed to be correct. By skipping assumed-valid blocks, joining nodes can avoid the costly signature verification of large portions of the blockchain at negligible security risks. However, joining nodes still download the complete blockchain to replay all historical transactions to create an up-to-date UTXO set.

Simple Block Pruning. To counter increasing storage requirements, Bitcoin users have the option to completely *prune raw blockchain data* [11] after a *full* initial synchronization. This step allows nodes to forget all historical blockchain data at the cost of its queryability. In contrast to local index pruning, block pruning is detrimental to the network health as block-pruning nodes are incapable of bootstrapping new nodes.

State-based Synchronization. While Bitcoin focuses on financial transactions, other cryptocurrencies, such as Ethereum [8], are also capable of executing smart contracts. Naturally, those cryptocurrencies have more complex state layouts as the full nodes need to keep track of every smart contract's state. Consequentially, Ethereum uses *Fast Sync* [28], which enables joining nodes to download a recent state and thereby avoids replaying all historical information. However, Ethereum still values the queryability of historical data, and thus joining nodes also download and persist all blocks, but do not have to process them during initial synchronization. In contrast to Bitcoin's proposal for Ultimate Compression, Fast Sync remains secure since Ethereum, by default, cryptographically ties its current state to each block [8]. Thus, nodes can verify the correctness of their obtained state directly via Ethereum's blockchain. Since other cryptocurrencies lack these header fields, Fast Sync is not immediately portable.

Takeaway. Developers have tackled the scalability issues of blockchain systems from different perspectives. However, all approaches have either limited efficiency, questionable security properties, are detrimental to network health, or are not portable to a variety of already deployed systems.

C. Proposed Block-Pruning Schemes

The insufficiency of post-deployment pruning schemes inspired various *alternative blockchain designs*, promising better scalability than established systems. We identify alternative designs that refine mere block-pruning schemes as well as designs proposing state-based or balance-based synchronization.

Simple Block Pruning. Palm et al. [29] present a distributed block-pruning scheme for established nodes in permissioned blockchains, i.e., blockchains jointly maintained by a fixed set of mutually known parties. A dedicated initiator defines a pruning algorithm that must be executed by all nodes to identify and prune now-irrelevant transactions in a way that all relevant data is still retrievable from other nodes. However, this approach focuses on permissioned blockchains and requires a dedicated initiator. Hence, the approach is inapplicable to public settings, which are open to unknown or unauthenticated parties, both for security and compatibility reasons.

State-based Synchronization. Similarly to Ethereum Fast Sync and inheriting its advantages and disadvantages, Rollerchain [14] proposes state-based initial synchronization. However, Rollerchain values performance over complete queryability, thereby significantly decreasing bootstrapping overhead as old information does not need to be transmitted and stored. Similarly, Marsalek et al. [30] propose a state-based synchronization based on Bitcoin but abandon compatibility by rejecting blocks that have invalid states attached.

Balance-based Synchronization. A special class of state-based block-pruning schemes simplifies the structure of what constitutes a state to allow for more efficient representations and updates [12], [13], [15], [31]. Typically, these schemes only keep track of existing accounts and their balances. The Mini-Blockchain scheme [12] replaces Bitcoin's UTXO set with an account tree that is cryptographically tied to each mined block. Joining nodes obtain the headerchain and a recent account tree to synchronize, before fully processing a tail of full blocks to preserve PoW-based security. However, such a scheme expects established nodes to compute slices of the recent account tree on demand, without commenting on the availability of all required data to rewind the account tree accordingly within the network. Mimblewimble [13] follows a similar approach, but emphasizes confidential transactions at the cost of synchronization performance as joining nodes have to obtain and verify rangeproofs for unspent funds [13]. Through their balance-based approach, both schemes limit the expressiveness of transactions. To overcome this limitation, Pascal [15] defines SafeBoxes as a replacement for mere account trees. SafeBoxes permit the generation of a limited number of accounts per block and are designed to enable upper-layer applications, but the limited availability of account spots is conceptually detrimental to network health. Finally, Vault [31] builds on Algorand [51] to enable the distribution of fragments of recent states across the network to reduce the per-node storage requirements. Therefore, Vault is inapplicable as an aid for existing, simpler cryptocurrencies.

Takeaway. Alternative blockchain designs have shown that incorporating cryptographic ties to recent state objects are a promising means to establish trust in state-based blockchain synchronization processes. However, extending existing systems with such capabilities immediately results in hard forks, which are difficult to deploy and thus highly debated [16].

V. Requirements for Secure Block Pruning

Blockchain systems require that sufficiently many nodes maintain a *full local copy* of the blockchain (cf. Section II). While this initial design becomes massively burdening for these nodes as well as joining nodes, multiple approaches to fully pruning historical data have been proposed (cf. Section IV-C). However, none of these approaches can be adapted to directly provide similar optimizations for established systems (e.g., Bitcoin) without provoking major incompatibilities. To realize *fully compatible* extensions for the network-wide pruning of obsolete information in existing cryptocurrencies, while *maintaining already established security levels*, we identify the following requirements and design goals:

(G1) Scalability. To be effective, pruning schemes must provide improvements for *all* metrics discussed in Section III, i.e., storage and bandwidth demands for joining and block-serving nodes, processing costs, and synchronization time.

(G2) Correctness. Starting from the initial genesis block, each joining node must obtain the same internal state, with or without the block-pruning scheme enabled, to ensure that the network's consensus about accepted transactions is kept intact. In particular, the node must learn about all accepted, non-obsolete events, and it must not accept any false events.

(G3) Verifiability. As security is a top priority, our pruning scheme must keep joining nodes able to verify the correctness of the synchronization process even in the presence of adversaries. Here, we require that the block-pruning scheme does not reduce the security of the overall blockchain system.

(G4) Compatibility. Popular and long-living blockchain systems are especially affected by scalability limitations. Instead of proposing new systems (cf. Section IV-C), all changes should be applicable to existing blockchains, especially Bitcoin, even during operation. Preferably, the scheme is opt-in, e.g., as achieved via velvet forks (cf. Section II).

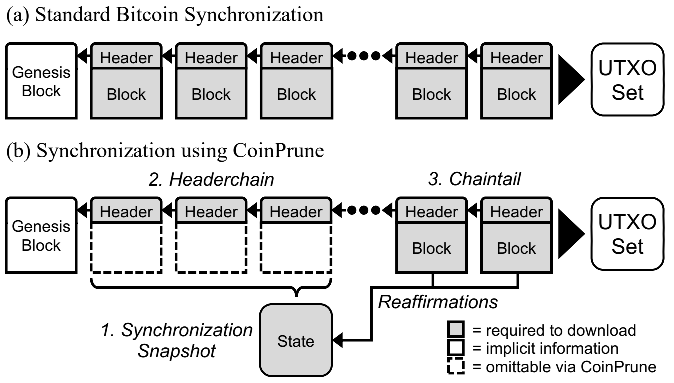
(a) Standard Bitcoin Synchronization
Genesis Block
Header
Block
UTXO Set
(b) Synchronization using CoinPrune
2. Headerchain
3. Chaintail
Reaffirmations
1. Synchronization Snapshot
State
= required to download
= implicit information
= omittable via CoinPrune

Fig. 1. High-level design overview of CoinPrune. Instead of downloading and verifying all blockchain data, joining nodes obtain a recent snapshot in a trustworthy manner due to its on-chain reaffirmations by multiple miners.

VI. CoinPrune Design

To address the challenges resulting from the ever-increasing size of existing blockchains, we present *CoinPrune*, our secure, state-based block-pruning scheme that is gradually deployable without protocol-breaking changes, e.g., to Bitcoin. After giving an overview of CoinPrune, we first present how nodes using our scheme coordinate via Bitcoin's blockchain and then present how joining nodes can bootstrap.

A. CoinPrune Overview

CoinPrune is designed to transfer scalability improvements of novel blockchain designs (cf. Section IV-C) to Bitcoin while keeping compatibility a priority. We now describe how *CoinPrune nodes*, i.e., Bitcoin nodes additionally supporting CoinPrune, jointly maintain recent snapshots on the blockchain, explain how new nodes can bootstrap securely using those snapshots, and outline benefits for the whole network.

Snapshot Maintenance. CoinPrune nodes periodically create *snapshots* of their current UTXO set. These snapshots are served to joining nodes instead of the entirety of historical blockchain data for reduced storage, bandwidth, and processing requirements **(G1)**. They are tied to the current *block height*, i.e., the position of the most recent block in the blockchain, and contain a well-ordered UTXO set for synchronization **(G2)** and verification **(G3)** purposes, respectively. To prevent malicious nodes from distributing incorrect snapshots, e.g., in an attempt to multiply their unspent funds, CoinPrune requires snapshots to be *publicly announced* to the blockchain by referencing a cryptographic *identifier* of each snapshot on-chain. CoinPrune-supporting miners place these announcements in their blocks' *coinbase transactions*, which miners issue to mint new coins. By utilizing an existing field in coinbase transactions, which may contain 100 B of arbitrary data, we keep CoinPrune Bitcoin-compatible **(G4)**. Other CoinPrune miners independently do the same, which causes nodes deriving snapshots from the same UTXO set to *mutually reaffirm* that snapshot's validity. This approach creates positive-only feedback, i.e., wrong snapshots are not rejected but tolerated and outpaced by valid reaffirmations given an honest majority of CoinPrune miners.

Bootstrapping Nodes. Instead of downloading all historical blockchain data, a joining node can securely bootstrap in three steps, as shown in Figure 1: First, the node obtains a *recent snapshot* either from its neighbors or through a snapshot-offering third party. If opting for P2P-based snapshot acquisition, the node downloads the snapshot that advocated by most neighbors, given an absolute majority for one snapshot. Second, the node downloads the *headerchain*, i.e., the interconnected and lightweight block headers, to learn about the blockchain branch with the most PoW in it. Third, the node downloads the *chaintail*, i.e., the full blocks starting from the snapshot's block height. Via the chaintail, the joining node can (a) catch up with recent transactions, and (b) inspect the full blocks for snapshot reaffirmations. If the joining node observes sufficiently many reaffirmations of the snapshot, it accepts the snapshot and concludes initial synchronization. Otherwise, the node starts over by discarding the insecure snapshot and reconnecting to a new set of neighbors.

Global Block Pruning. Since joining nodes can securely bootstrap from the headerchain, the snapshot, and the chaintail, *all* CoinPrune nodes may now *safely prune historical blocks* prior to the snapshot. As new snapshots are reaffirmed periodically, nodes may also prune aging snapshots as well without hurting the network health. Single *archival nodes* may still keep a full blockchain copy to retain full and reliable queryability of historical data.

B. Adapted Data Management

To understand CoinPrune in detail, we discuss the layout of its snapshots and the required changes to nodes' local data management stemming from the pruning of historical blocks.

Snapshot Creation. Each snapshot corresponds to a specific block height, meaning that it represents a serialization of the UTXO set obtained from processing all blocks up to and including that height. A CoinPrune snapshot consists of a simple header and multiple chunks of serialized UTXOs, and it is referenced on-chain by a cryptographic identifier. The header holds the snapshot's corresponding block height, that block's identifier, and the number of chunks in the snapshot. The identifier is a special hash value created over the snapshot's header and chunks to uniquely represent the snapshot in a succinct manner. First, the header and each chunk are hashed individually using Bitcoin's HASH256 function (SHA256 applied twice). Then, the snapshot identifier is the hash value of the concatenation of these hash values. Using this simple snapshot serialization, joining nodes are immediately aware of all available chunks and can independently request individual chunks from different neighbors in parallel. Further, we limit chunk sizes to 1 MB akin to Bitcoin's maximum block size.

Persisted Information. By shifting to snapshot-based synchronization, nodes may now prune historical full blocks. However, the nodes must remain capable of serving the full headerchain to joining nodes. Before pruning blocks, these nodes thus need to persist some information currently held by Bitcoin's block index. These are block identifiers, headers, block heights, the amount of PoW, the number of transactions, and the block's timestamp. Persisting this data, a recent snapshot, and the chaintail of not-yet-prunable full blocks is now sufficient to securely bootstrap joining nodes.

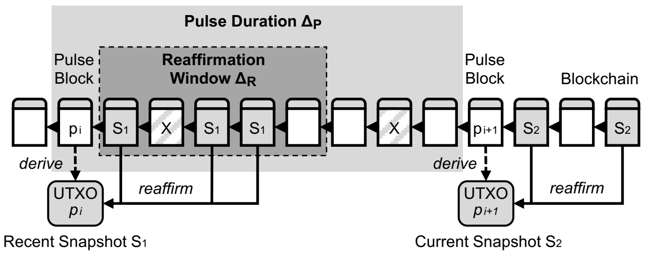

Fig. 2. Our pulse-based coordination triggers the creation of new snapshots. Any invalid or delayed reaffirmations are ignored.

C. Coordination of CoinPrune Nodes

CoinPrune relies on an honest majority of snapshot-serving nodes to mutually reaffirm recent snapshots' correctness to provide a trust anchor for joining nodes. To this end, these established nodes must agree on when to create snapshots and how to publish reaffirmations. As shown in Figure 2, CoinPrune uses *pulse blocks* p_i issued in constant intervals Δ_P, e.g., every 10 000 blocks, and synchronize the snapshot creation and reaffirmation activities among CoinPrune nodes. Through Δ_P, CoinPrune can tune the frequency of snapshot creation and thereby adjust processing and storage overheads for CoinPrune nodes. Nodes create a new snapshot whenever a pulse block has been mined. They base the snapshot on the pulse block's corresponding UTXO set. Subsequently, all CoinPrune miners reaffirm the new snapshot in blocks they mine until the shorter *reaffirmation window* Δ_R expires. To reaffirm a snapshot, the miners add the snapshot's identifier to their blocks' coinbase fields. While not all Bitcoin blocks mined during Δ_R must contain a reaffirmation and individual blocks may reaffirm invalid snapshots, an honest majority among CoinPrune miners ensures that the valid, new snapshot will accumulate reaffirmations the fastest. Nodes *accept* the snapshot with the most reaffirmations if that snapshot was reaffirmed at least k times during Δ_R. This acceptance threshold k protects nodes from accepting snapshots reaffirmed by individual adversaries with low mining power in times of low overall CoinPrune participation. Consequently, all nodes can safely ignore any reaffirmations that are outpaced by reaffirmations of another snapshot or that lie outside of Δ_R. If no snapshot reaches k reaffirmations during Δ_R, this pulse is invalid, and pruning is delayed until the next pulse starts.

D. Bootstrapping New Nodes

The reaffirmations periodically published on Bitcoin's blockchain allow joining nodes to bootstrap as follows. First, the node obtains a recent snapshot. The node can either acquire a recent snapshot through external means, e.g., mirror servers, or ask its neighbors for the most recent snapshots they are aware of using off-chain P2P requests. Second, the joining node downloads and verifies the headerchain from its neighbors to learn about the blockchain branch with the most PoW in it, as is already done by Bitcoin [25]. Third, instead of downloading and processing all historical data, the joining node applies the previously obtained snapshot in good faith to initially fill its UTXO set. Finally, the joining node fetches and processes the *chaintail*, i.e., the remaining full blocks succeeding the snapshot's block height, to finalize synchronizing its UTXO set. During this full synchronization phase, the joining node additionally inspects the chaintail's coinbase transactions for reaffirmations of its applied snapshot. If the node learns that its snapshot was the most-reaffirmed one during Δ_R and was reaffirmed at least k times, it accepts the snapshot, which concludes the bootstrapping step. Otherwise, the joining node aborts and obtains a different snapshot from another source, e.g., by connecting to a new set of neighbors.

VII. Seamless Integration into Bitcoin

CoinPrune's main feature is its immediate applicability to Bitcoin **(G4)**. In this section, we present our means to achieve *gradual opt-in deployability* to Bitcoin via a velvet fork, assuming that a sufficient share of honest miners makes a rational choice to support CoinPrune, e.g., to preserve storage.

On-Chain Data. Although snapshot reaffirmations must be publicly announced on Bitcoin's blockchain, CoinPrune's utilization of only a block's coinbase field prevents any protocol-breaking changes. Full nodes that are unaware of CoinPrune will ignore any snapshot reaffirmation, and CoinPrune nodes will never reject blocks containing incorrect reaffirmations. Instead, they will try to outpace incorrect reaffirmations with legitimate ones. Hence, our scheme fulfills the requirements for a gradually deployable velvet fork [17], [18]. To prevent CoinPrune nodes from confusing snapshot reaffirmations with other coinbase data, we propose to encapsulate the reaffirmation accordingly, e.g., using a unique prefix and separators such as `CoinPrune/[snapshot_id]/`.

Peer-to-Peer Protocol. Even though CoinPrune allows for external snapshot sources, most joining nodes will likely rely on Bitcoin's network to obtain their initial snapshot. To enable this Bitcoin-intrinsic snapshot acquisition, we extend Bitcoin's P2P protocol [52] with an additional `GETSTATE` message type sent by CoinPrune nodes. Joining nodes send a `GETSTATE` message to each neighbor to learn about available recent snapshots. Each neighbor responds with an inventory (`INV` message) that contains the hash values of the snapshot header and the chunks of their most recent available and successfully reaffirmed snapshot as `STATE` objects. The joining node uses these `INV` messages to determine which snapshot to obtain and to derive that snapshot's identifier. Then, the node continues to request individual chunks of the most-advertised snapshot from its neighbors using sequences of `GETDATA` messages. Finally, the node applies the state once all chunks are available. For increased compatibility, we restrict chunk sizes to 1 MB, i.e., Bitcoin's maximum block size, and we introduce a new service flag for Bitcoin's `VERSION` handshake to avoid sending unknown messages to CoinPrune-unaware nodes.

Takeaway. Bitcoin nodes can adopt CoinPrune immediately without creating forks or causing issues on the P2P layer.

VIII. Security Discussion

We argue that CoinPrune bootstraps nodes correctly **(G2)** based on verifiable snapshots **(G3)** as (a) our on-chain reaffirmations reliably reference snapshots from arbitrary sources, that (b) positive-only feedback from an honest majority of CoinPrune miners establishes trust in snapshots, and that (c) CoinPrune is not prone to additional P2P-layer attacks.

Verifying Snapshot Validity. Joining nodes must be able to verify the benignity of a snapshot reliably based on our on-chain reaffirmations. To achieve this, CoinPrune derives snapshot identifiers from a cryptographic hash function in a layered manner. Each snapshot chunk is hashed individually. Hence, full nodes cannot alter individual chunks during initial synchronization without having the joining node notice the manipulation. Furthermore, the snapshot identifier covers the snapshot's meta information as well as all chunks' hash values. Hence, joining nodes can verify that they obtain exactly the required snapshot chunks, and they know precisely to which block height the snapshot corresponds. Consequently, no adversary can trick a joining node into accepting an altered snapshot, nor can they create inconsistencies as to how to apply the snapshot (e.g., to reintroduce obsolete UTXOs).

Reliability of Reaffirmations. Even though joining nodes can reliably associate their obtained snapshots with on-chain reaffirmations, our scheme deliberately tolerates the presence of reaffirmations to invalid snapshots to maximize CoinPrune's compatibility with Bitcoin. Consequentially, we need to prevent joining nodes from accepting invalid snapshots as sufficiently reaffirmed. CoinPrune achieves this in a similar manner as Bitcoin keeps its blockchain secure. Assuming an honest majority among CoinPrune-supporting miners, snapshots that are derived correctly from pulse blocks will eventually accumulate reaffirmations faster than other snapshots. Hence, joining nodes should not rely on snapshots with only fewer than k on-chain reaffirmations. We note that low adoption of CoinPrune could give slow-but-steady malicious miners a relative advantage. However, by introducing the reaffirmation window Δ_R, adversaries with low relative mining power within the Bitcoin network are highly unlikely to successfully reaffirm an invalid snapshot in time even if they temporarily constitute a dishonest majority among CoinPrune nodes. Finally, a trusted third party can temporarily aid this transition phase by releasing signed snapshot identifiers redundantly to on-chain reaffirmations.

P2P Attacks. CoinPrune integrates well with Bitcoin's P2P protocol and only adds `GETSTATE` messages and `STATE` inventories. Hence, considerations regarding Bitcoin's resilience against DoS attacks or Eclipse attacks [53] directly transfer to CoinPrune. Furthermore, adversaries cannot partially manipulate or completely replace valid snapshots, as discussed above. If joining nodes thus observe an attempted attack, they can abort bootstrapping and connect to a new set of neighbors.

Takeaway. CoinPrune enables joining nodes to obtain snapshots from any source and to still rely on its on-chain reaffirmations to synchronize correctly with the Bitcoin network.

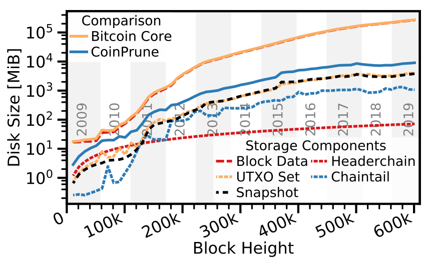

Fig. 3. Currently, our scheme already reduces storage requirements by two orders of magnitude.

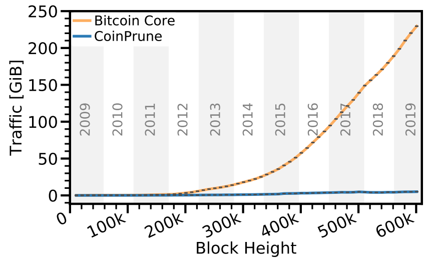

Fig. 4. CoinPrune allows bootstrapping with vastly reduced amounts of traffic, unburdening all nodes.

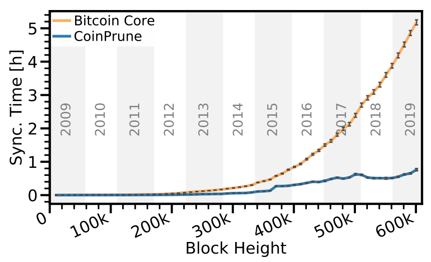

Fig. 5. Our scheme reduces initial synchronization times, yet the snapshot size impacts performance.

IX. Performance Evaluation

We now demonstrate that CoinPrune enables massive performance savings for Bitcoin nodes **(G1)**. After describing our testbed setup, we present the storage savings achieved for all nodes. Further, we show that traffic and synchronization time for joining nodes are massively reduced as well.

A. Testbed Setup for Synchronization Measurements

We created a proof-of-concept implementation of CoinPrune based on Bitcoin Core v0.17.1. Our measurements run on a server (2× Intel Xeon E5-2630 v4, 32 GB RAM, 8 TB Seagate IronWolf ST8000VN0022-2EL112), which synchronizes from eight identical commodity PCs (Intel Core2 Quad Q9400, 8 GB RAM, 500 GB Hitachi Deskstar 7K500) via a Linksys SLM2024 Gigabit switch. We measure synchronizing with Bitcoin's blockchain in increments of 10 000 blocks up until a block height of 600 000 (Oct 19, 2019) and additional 1000 blocks (∼1 week of blocks) as our chaintail. We perform synchronization via vanilla Bitcoin Core in one go and start synchronization via CoinPrune from the snapshots' respective block heights. While storage requirements are fully determined by the blockchain data, synchronization times and traffic may vary. Hence, they were averaged over ten independent runs, and we show the 99 % confidence intervals. We omitted to check the coinbase field for the snapshot identifier to be able to use Bitcoin's real blockchain for our measurements.

B. Storage Savings

CoinPrune allows all Bitcoin nodes to prune historical blocks in exchange for maintaining a recent snapshot and the headerchain to serve joining nodes. In Figure 3, we depict how the main contributors to Bitcoin's storage demand changed over time in comparison to the serialized snapshot and headerchain required to operate CoinPrune. From this, we derive the overall storage requirements for Bitcoin Core and CoinPrune, respectively. For Bitcoin's storage requirements, we consider the heavily dominating `blocks` folder containing raw block data, information required to rewind blocks efficiently, and the block index, as well as the `chainstate` folder holding the UTXO set. In contrast to this, CoinPrune needs to store one serialized snapshot and the serialized headerchain, as well as the UTXO set and chaintail for live operation. Our measurements show that the sizes of serialized snapshots align well with those of Bitcoin's UTXO set. Minor variances stem from different encodings of both data structures. Further, persisting the headerchain to reconstruct Bitcoin's block index comes at only negligible costs of 125.00 B per block, resulting in a headerchain size of 71.53 MiB for our latest measurement. Finally, considering block heights starting from 300 000, the chaintail has an average size of 0.99 GiB. Overall, CoinPrune nodes could thus historically reduce their storage requirements by 85.60 %, with the largest absolute and relative savings, currently 255.42 GiB, at higher block heights. These savings account for a decrease of two orders of magnitude, with the potential for becoming even larger as the blockchain grows.

C. Evaluation of Synchronization Performance

Pruning obsolete data not only relieves Bitcoin nodes from storage depletion but also joining nodes benefit from widespread adoption of CoinPrune. As shown in Figure 4, the reduced storage requirements directly translate to a reduction in traffic required to synchronize with the Bitcoin network. For instance, synchronizing from a snapshot on block height 600 000 with a chaintail length of 1000, joining nodes only inflict 5.03 GiB of traffic when using CoinPrune, whereas legacy nodes would cause 229.62 GiB of traffic to bootstrap successfully. Over the whole blockchain, achievable savings average at 92.50 %. Joining nodes currently have to obtain two orders of magnitude less data during initial synchronization, which is largely dominated by acquiring the snapshot.

A similar trend can be observed for the overall synchronization time of joining nodes, i.e., obtaining and verifying the headerchain, the snapshot, and the chaintail. Figure 5 shows that CoinPrune improves synchronization times over Bitcoin's whole history, resulting in savings of 75.58 % on average for joining nodes. Even though Bitcoin mitigates reverifying very old transactions due to its assumed-valid blocks (cf. Section IV-B), joining nodes still must replay the whole transaction graph. Contrarily, CoinPrune avoids this step as well due to its reliance on snapshots. In consequence, CoinPrune currently enables joining nodes to catch up with the Bitcoin network in 46 min instead of 5 h using standard Bitcoin. This time saving is especially beneficial as initial synchronization is often considered a major scalability concern [9], [19].

Takeaway. The snapshot-based approach of CoinPrune unburdens both established and joining nodes from major overhead stemming from Bitcoin's bootstrapping process regarding storage, traffic, and synchronization time. Hence, CoinPrune establishes a secure and effective means to vastly improve Bitcoin's long-term durability.

X. Conclusion

CoinPrune tackles the increasing scalability issues of public blockchain systems such as Bitcoin. These issues stem from growing storage and performance issues as nodes keep, redistribute, and reverify obsolete, historical data for security reasons. We have shown that we can extend Bitcoin with an effective and secure block-pruning scheme without protocol-breaking changes by having honest miners create snapshots of Bitcoin's UTXO set and mutually reaffirm their correctness on the blockchain. Averaged over Bitcoin's lifetime, joining nodes could have reduced synchronization times by 76 % this way. Furthermore, all nodes are relieved of keeping historical data, reducing Bitcoin's current storage requirements of roughly 264 GiB to 9 GiB, saving currently 255 GiB with the prospects of even more saving potential as the blockchain grows.

As future work, we plan to further tackle current limitations of CoinPrune and block pruning in general, e.g., preserving the currently over 5.3 m application-specific `OP_RETURN` outputs, the efficiency of snapshot creation, and the potential for removing illicit content from the UTXO set. Hence, CoinPrune has the potential to secure Bitcoin's long-term scalability in an immediately applicable, but gradually deployable way.

Acknowledgements. This work has been funded by the German Federal Ministry of Education and Research (BMBF) under funding reference numbers 16KIS0443, 16DHLQ013, and Z31 BMBF Digital Campus. The funding under reference number Z31 BMBF Digital Campus has been provided by the German Academic Exchange Service (DAAD). The responsibility for the content of this publication lies with the authors.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," White paper, 2008.

[2] "Proof of Existence," accessed 2020-04-11. https://proofofexistence.com

[3] "Namecoin," accessed 2020-04-11. https://namecoin.org

[4] M. Henze, B. Wolters *et al.*, "Distributed Configuration, Authorization and Management in the Cloud-based Internet of Things," in *IEEE TrustCom/BigDataSE/ICESS*, 2017.

[5] M. Chase and S. Meiklejohn, "Transparency overlays and applications," in *ACM CCS*, 2016.

[6] K. Nikitin, E. Kokoris-Kogias *et al.*, "CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds," in *USENIX Security*, 2017.

[7] R. Matzutt, J. Pennekamp *et al.*, "Utilizing Public Blockchains for the Sybil-Resistant Bootstrapping of Distributed Anonymity Services," in *ACM ASIACCS*, 2020.

[8] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," White paper, 2016.

[9] K. Croman, C. Decker *et al.*, "On Scaling Decentralized Blockchains," in *IFCA FC Bitcoin Workshop*, 2016.

[10] Blockchain.com, "Blockchain Charts," 2011, accessed 2020-04-11. https://www.blockchain.com/charts

[11] Bitcoin Project, "Bitcoin Core version 0.11.0 released," 2015, accessed 2020-04-11. https://bitcoin.org/en/release/v0.11.0

[12] J. D. Bruce, "The Mini-Blockchain Scheme," White paper, 2014.

[13] A. Poelstra, "Mimblewimble," White paper, 2016.

[14] A. Chepurnoy, M. Larangeira, and A. Ojiganov, "Rollerchain, a Blockchain With Safely Pruneable Full Blocks," White paper, 2016.

[15] H. Schoenfeld and A. Molina, "Pascal: An Infinitely Scalable Cryptocurrency," White paper, 2019.

[16] D. Morgan, "The Great Bitcoin Scaling Debate – A Timeline," 2017, accessed 2020-04-11. https://hackernoon.com/the-great-bitcoin-scaling-debate-a-timeline-6108081dbada

[17] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof-of-work," *IACR Cryptology ePrint Archive*, vol. 2017/963, 2017.

[18] A. Zamyatin, N. Stifter *et al.*, "A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice," in *IFCA FC Bitcoin Workshop*, 2018.

[19] J. Lopp, "Bitcoin Full Validation Sync Performance," 2018, accessed 2020-04-11. https://blog.keys.casa/bitcoin-full-validation-sync-performance

[20] Bitcoin Project, "Hot Wallet," 2012, accessed 2020-04-11. https://en.bitcoin.it/wiki/Hot_wallet

[21] Bitcoin Project, "Lightweight Node," 2018, accessed 2020-04-11. https://en.bitcoin.it/wiki/Lightweight_node

[22] A. Reiner, "Ultimate Blockchain Compression w/ Trust-free Lite Nodes," 2012, accessed 2020-04-11. https://bitcointalk.org/index.php?topic=88208

[23] Bitcoin Project, "Bitcoin-Qt version 0.8.0 released," 2013, accessed 2020-04-11. https://bitcoin.org/en/release/v0.8.0

[24] Bitcoin Project, "Bitcoin Core version 0.15.0 released," 2017, accessed 2020-04-11. https://bitcoin.org/en/release/v0.15.0

[25] Bitcoin Project, "Bitcoin Core version 0.10.0 released," 2015, accessed 2020-04-11. https://bitcoin.org/en/release/v0.10.0

[26] S. Nakamoto, "Bitcoin 0.3.2 released," 2010, accessed 2020-04-11. https://bitcointalk.org/index.php?topic=437

[27] Bitcoin Project, "Bitcoin Core version 0.14.0 released," 2017, accessed 2020-04-11. https://bitcoin.org/en/release/v0.14.0

[28] P. Szilágyi, "eth/63 fast synchronization algorithm," 2015, accessed 2020-04-11. https://github.com/ethereum/go-ethereum/pull/1889

[29] E. Palm, O. Schelén, and U. Bodin, "Selective Blockchain Transaction Pruning and State Derivability," in *IEEE CVCBT*, 2018.

[30] A. Marsalek, T. Zefferer *et al.*, "Tackling Data Inefficiency: Compressing the Bitcoin Blockchain," in *IEEE TrustCom/BigDataSE*, 2019.

[31] D. Leung, A. Suhl *et al.*, "Vault: Fast Bootstrapping for the Algorand Cryptocurrency," in *ISOC NDSS*, 2019.

[32] D. Ron and A. Shamir, "Quantitative Analysis of the Full Bitcoin Transaction Graph," in *IFCA FC*, 2013.

[33] S. Meiklejohn, M. Pomarole *et al.*, "A Fistful of Bitcoins: Characterizing Payments Among Men with No Names," in *ACM IMC*, 2013.

[34] M. Bartoletti and L. Pompianu, "An analysis of Bitcoin OP_RETURN metadata," in *IFCA FC Bitcoin Workshop*, 2017.

[35] R. Matzutt, J. Hiller *et al.*, "A Quantitative Analysis of the Impact of Arbitrary Blockchain Content on Bitcoin," in *IFCA FC*, 2018.

[36] A. Sward, I. Vecna, and F. Stonedahl, "Data Insertion in Bitcoin's Blockchain," *Ledger*, vol. 3, 2018.

[37] S. Delgado-Segura, C. Pérez-Solà *et al.*, "Analysis of the Bitcoin UTXO set," in *IFCA FC Bitcoin Workshop*, 2018.

[38] G. Ateniese, B. Magri *et al.*, "Redactable Blockchain – or – Rewriting History in Bitcoin and Friends," in *IEEE EuroS&P*, 2017.

[39] I. Puddu, A. Dmitrienko, and S. Capkun, "μchain: How to forget without hard forks," *IACR Cryptology ePrint Archive*, vol. 2017/106, 2017.

[40] R. Matzutt, M. Henze *et al.*, "Thwarting Unwanted Blockchain Content Insertion," in *IEEE IC2E Workshops*, 2018.

[41] D. Deuber, B. Magri, and S. A. K. Thyagarajan, "Redactable Blockchain in the Permissionless Setting," in *IEEE S&P*, 2019.

[42] M. Florian, S. Henningsen *et al.*, "Erasing Data from Blockchain Nodes," in *IEEE EuroS&PW*, 2019.

[43] A. Dorri, S. S. Kanhere, and R. Jurdak, "MOF-BC: A Memory Optimized and Flexible Blockchain for Large Scale Networks," *Future Gener. Comput. Syst.*, vol. 92, 2019.

[44] L. Luu, V. Narayanan *et al.*, "A Secure Sharding Protocol For Open Blockchains," in *ACM CCS*, 2016.

[45] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: Scaling Blockchain via Full Sharding," in *ACM CCS*, 2018.

[46] E. Kokoris-Kogias, P. Jovanovic *et al.*, "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding," in *IEEE S&P*, 2018.

[47] J. Poon and T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," White paper, 2016.

[48] M. Green and I. Miers, "Bolt: Anonymous Payment Channels for Decentralized Currencies," in *ACM CCS*, 2017.

[49] Bitcoin Project, "Bitcoin Core version 0.12.0 released," 2016, accessed 2020-04-11. https://bitcoin.org/en/release/v0.12.0

[50] Bitcoin Project, "Bitcoin Core version 0.13.0 released," 2016, accessed 2020-04-11. https://bitcoin.org/en/release/v0.13.0

[51] Y. Gilad, R. Hemo *et al.*, "Algorand: Scaling Byzantine Agreements for Cryptocurrencies," in *ACM SOSP*, 2017.

[52] Bitcoin Project, "P2P Network," 2017, accessed 2020-04-11. https://bitcoin.org/en/p2p-network-guide

[53] E. Heilman, A. Kendler *et al.*, "Eclipse Attacks on Bitcoin's Peer-to-Peer Network," in *USENIX Security*, 2015.