On the Storage Overhead of Proof-of-Work Blockchains

Alessandro Sforzin
NEC Laboratories Europe

Matteo Maso
AiSight

Claudio Soriente
NEC Laboratories Europe

Ghassan Karame
Ruhr-University Bochum

***Abstract*—Permissionless blockchains such as Bitcoin have long been criticized for their high computational and storage overhead. Unfortunately, while a number of proposals address the energy consumption of existing Proof-of-Work deployments, little attention has been given so far to remedy the storage overhead incurred by those blockchains. In fact, it seems widely acceptable that full nodes supporting the blockchains have to volunteer hundreds of GBs of their storage, to store and verify all transactions exchanged in the system.**

In this paper, we explore the solution space to effectively reduce the storage footprint of Proof-of-Work based blockchains. To do so, we analyze, by means of thorough empirical measurements, how existing full blockchain nodes utilize data from the shared ledger to validate incoming transactions/blocks. Based on this analysis, we show that it is possible for full nodes to locally reduce their storage footprint to approximately 15 GB, without any modification to the underlying protocol. We also discuss other client-side strategies to further reduce the storage footprint while incurring negligible computational overhead on the nodes.

I. Introduction

Blockchains are receiving increasing attention among researchers and practitioners, owing to their promise to efficiently manage business processes in a decentralized manner. Although the literature features a large number of blockchains that notably differ in their leader election and consensus protocols, almost all existing blockchains rely on the presence of a shared ledger that enables blockchain nodes to agree on the order and correctness of information (i.e., transactions and blocks).

Permissionless blockchains (such as Bitcoin and Ethereum) have been heavily criticized due to their high computational and storage overhead. At the time of writing, Bitcoin—arguably the most popular instantiation of a permissionless blockchain—incurs an estimated annual energy consumption of 105.70TWh (see [8]), and needs more than 370 GB of space to store the ledger.

While the computational overhead in existing blockchains can be remedied by replacing Proof-of-Work with newer, more energy-friendly protocols such as Proof-of-Stake or Byzantine-Fault-Tolerant protocols, little work exists to remedy the storage overhead incurred by today's blockchains.

A high storage overhead is mostly evident in those blockchains that have witnessed the largest adoption. That is, the larger is the adoption of a given blockchain platform, the more transactions that are exchanged, and in turn the bigger is the storage required to maintain the shared ledger. On the one side, keeping all transaction data is essential to ensure the security of the system and to make sure that no transaction is spent more than once. On the other side, storing hundreds of GBs, is one of the main reasons why many users shy away from running full nodes (i.e., nodes that store the full ledger).

Previous work has proposed a number of fixes and extensions to mitigate the storage overhead of existing blockchain platforms [10], [13], [14], [16]. Most proposals introduce protocols to create and maintain periodic checkpoints (or snapshot) of the ledger, so that data belonging to previous snapshots can be safely deleted. However, implementing a checkpointing strategy require either to modify the way the blockchain works or a fork.

In terms of deployed solutions, Bitcoin allows users to prune the blockchain storage by defining a threshold (in GB or in block height) below which content in their local copy is trimmed [3]. Such pruning techniques are available in Bitcoin clients and can be locally used as a stand-alone, independent solution to reduce the storage overhead of the blockchain. However, the choice of the threshold value (either in terms of disk space or in terms of block height) is left to the user without any guidelines. On the one hand, storing too many blocks may not be feasible for all clients and would cause unnecessary data to be stored on disk. On the other hand, storing too few blocks would result in the deletion of data that may still be necessary to verify unspent transactions. Note that verification of transactions for which data has been deleted, incurs in additional communication overhead to fetch the required data from the network.

In this paper, we address the problem of efficiently managing the storage overhead incurred by existing blockchains and present the first study on how blockchain nodes use data from the shared ledger to validate transactions and blocks. We focus on Bitcoin since (i) it is arguably the most popular blockchain where the storage problem is most relevant, and (ii) publicly available data allows us to study the behavior of the network over a fairly long time period and obtain meaningful insights towards reducing the storage footprint. To do so, we start by analyzing, by means of thorough empirical measurements, how existing Bitcoin nodes manage data from the shared ledger to validate incoming transactions/blocks. Based on these findings, we explore the solution space to effectively manage the storage of existing PoW blockchains. To this end, we adapted a blockchain parser based on [5]

This work was supported in part by the by the European Commission H2020 TeraFlow Project under Grant Agreement No 101015857.

to compute the storage savings of the various strategies we devise.

Unlike common beliefs, our results show that it is possible for full Bitcoin nodes to locally reduce their storage footprint by approximately 95.9% without any modification to the underlying protocol and with no appreciable overhead to validate transactions. Moreover, we show that an archival node—wishing to store all information in the blockchain without any loss—could save up to 29% of storage space without losing any information from the ledger. This results in 5-10% more storage savings when compared to existing compression algorithms that can achieve a maximum compression rate of up to 24% on the Bitcoin ledger—and without requiring the heavy computational load associated with (de-)compression. Our parser will be released as open-source to better aid the community in estimating the actual storage needs of Bitcoin nodes as the ledger grows in size. We stress at this point that our observations are not restricted to Bitcoin and equally apply to the myriad of altcoins (or forks of the Bitcoin blockchain) that are currently deployed (e.g., Dogecoin, Bitcoin Cash, Litecoin, Monacoin).

The remainder of the paper is organized as follows. In Section II, we overview the storage requirements in existing blockchains and discuss related work in the area. In Section III, we empirically measure the transaction age and the storage overhead incurred in current Bitcoin transactions. In Section IV, we explore the space of possible solutions that may allow a Bitcoin node to reduce the ledger's footprint on its local storage and we evaluate the effectivness of those strategies in Section V. Finally, we conclude the paper in Section VI.

II. Background & Related Work

In this section, we introduce relevant background on Proof-of-Work (PoW) blockchains, with a focus on transactions validation and storage.

A. The Need for Storage in Existing Blockchains

PoW-based blockchains leverage Proofs of Work (PoW) as a public timestamping mechanism in order to prevent double-spending attacks. In practice, transactions are broadcasted and special nodes called *miners* add those transactions that they consider valid in a so-called *block*. A block is valid only if it contains valid transactions and the solution to a cryptographic puzzle. New blocks are cryptographically tied to previous ones via hash chains and, even if different chains of blocks can co-exist, only the longest chain is considered valid. In a nutshell, only transactions included in the blocks of the longest chain are considered valid. A miner that outputs a new block, broadcasts it so that other nodes can check its validity by checking the validity of its transactions and the correctness of the solution to the cryptographic puzzle. Nodes that are not mining (i.e., that do not contribute to block creation) are called full nodes. These nodes verify all exchanged information (blocks and transactions) in the blockchain and therefore have to store the full blockchain ledger[1]—albeit without any explicit incentives.

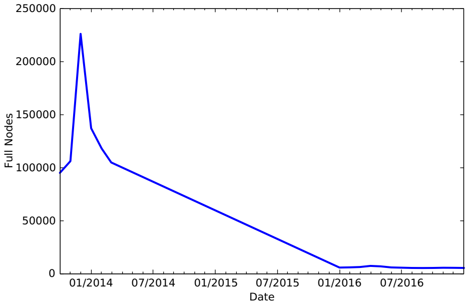

Fig. 1. Evolution of the number of Bitcoin full nodes over time. Estimates are adapted from [12].

For example, at the time of writing, a miner that succeeds in mining a block receives a fixed revenue of 6.25 BTCs and a variable profit comprising of all the fees that are included in the confirmed transactions. Full nodes are not rewarded by any means in spite of their critical role to preserve the security of the whole system.

Given the huge adoption of PoW-based blockchains [2], the storage requirements on full nodes has considerably increased. For instance, in 2014, the Bitcoin ledger was approximately 15 GBs. Throughout 2021, Bitcoin's ledger grew to approximately 371 GBs. In turn, as shown in Figure 1, the number of full Bitcoin nodes (that store the full ledger) dropped from 200,000 in 2014 to approximately 5,000 in 2016. Given lack of incentives, this number is only expected to decrease in the future.

We note that the majority of blockchain platforms also allow nodes that act as "lightweight clients". A lightweight client—usually a device with limited resources such as a smartphone—only downloads and verifies a small part of the chain. For example, the Bitcoin community provides the BitcoinJ[3], PicoCoin[4] and Electrum[5] clients implementing the Simple Payment Verification (SPV) mode [15], where the clients connect to a full node that has access to the complete blockchain and can help the client to confirm transactions. While lightweight clients ease the adoption of blockchain technology, we argue that full nodes are essential for a blockchain platform to thrive and solutions to minimize the burden (including the storage burden) of full nodes are necessary.

B. Transactions & Scripts

Many permissionless blockchains (such as Bitcoin) rely on the concept of Unspent Transaction Output (UTXO) to capture transactions in the platform. Essentially, a transaction is defined by a set of inputs (TxIn) and outputs (TxOut) that

[1] Without this information, an adversary can perform history corruption attacks—effectively presenting another chain of blocks (and the transactions therein) as the "main" chain.

[2] For instance, Bitcoin processes around 2000 transactions every 10 minutes at the time of writing.

[3] https://bitcoinj.org/

[4] https://github.com/jgarzik/picocoin

[5] https://electrum.org/

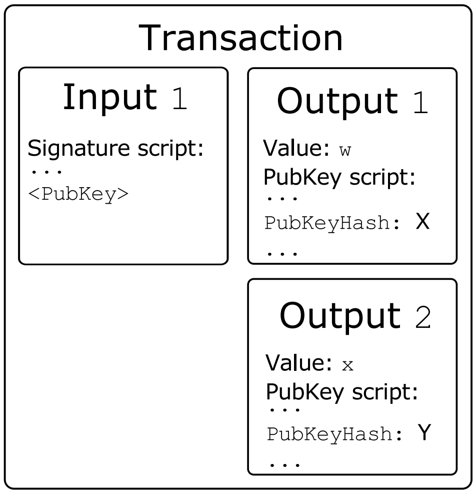

Fig. 2. Example of a Bitcoin transaction with 1 input and 2 outputs.

dictate the transfer the ownership of a sets of coins, specified in the TxIns, to a set of Bitcoin addresses, each of which get the amount of coins specified in the TxOuts. Concretely, each TxIn refers to one or more TxOuts of older transactions.

A transaction is invalid if the value of its TxOuts exceeds the value of its TxIns. However, if the TxIns value exceeds the value of the TxOuts, the miner who outputs the block storing the transaction can claim the difference as a transaction fee. The exact conditions under which an output can be spent are encoded with a set of scripts, and only the participants that are able to provide the correct input to the script, such that it evaluates to true upon execution, are allowed to spend the coins output by a given Bitcoin transaction.

Scripts refer to a custom non-Turing complete scripting language that are designed with the aim to support different types of transactions and extend the applicability of transaction beyond the simple transfer of funds. Scripts are stack-based, support a number of functions (commonly referred to as opcodes), and either evaluate to true or false. The language supports dozens of different opcodes ranging from simple comparison opcodes to cryptographic hash functions and signature verification. Since scripts are supposed to be executed by all blockchain node, they could be abused to conduct denial-of-service attacks; therefore, a considerable number of opcodes have been temporarily disabled. This was one of the main reasons why scripts do not provide rich support when compared to standard programming languages. The most common type of scripts found in Bitcoin's historical data are *pay-to-pubkey-hash* (P2PKH), *pay-to-pubkey* (P2PK), and *pay-to-script-hash* (P2SH).

Figure 2 depicts a simplified transaction with one input and two outputs. In this example, the transaction spends w BTCs to address X and x BTCs to address Y. The outputs that have not yet been spent (i.e., the two outputs of the transaction), are commonly referred to as unspent transaction outputs (UTXO).

Bitcoin keeps an up-to-date database of UTXOs, which it updates by adding or removing TxOuts created, or spent, by new transactions.

C. Existing methods to shrink storage

As mentioned earlier, little work has addressed the problem of reducing the storage footprint of blockchain nodes..

Florian et al., [11] suggest that UTXOs can be deleted and space can be saved if one is willing to trust other nodes to verify the validity of transactions including UTXOs that have been locally erased—essentially, the node acts as a lightweight client if the transaction to be validated includes at least one UTXO that has been deleted.

A number of proposals suggest to limit the size of the data to be stored by using an account-based model where the system keeps track only of those accounts that have a positive balance [9], [16]. Other proposals [9], [10], [13], [14] introduce extensions to existing systems to create and maintain periodic snapshots (or checkpoints) of the ledger, so that data belonging to previous snapshots can be safely deleted.

Real-world solutions focus on "pruning". For instance, pruning in Bitcoin (and other altcoins) was introduced with Bitcoin Core v0.11 [3] in 2015. Concretely, nodes can set a flag to specify the amount of disk space that Bitcoin can use for blocks and data, starting from a minimum of 550 MB (288 blocks, about two days worth of blocks). However, the choice of the threshold value (either in terms of disk space or in terms of block height) is left to the user without any guidelines. On the one hand, storing too many blocks may not be feasible for all clients and would cause unnecessary data to be stored on disk. On the other hand, storing too few blocks would result in the deletion of data that may still be necessary to verify unspent transactions. Such nodes can neither relay missing blocks, nor maintain transaction indexes anymore. Note that verification of transactions for which data has been deleted, incurs in additional communication overhead to fetch the required data from the network.

More recently, blockchains such as Bitcoin introduced the segregated witness (segwit) structure, which stores data required to validate transactions, that is, scripts and signatures, outside of the associated blocks [4]. While this solution reduces the communication overhead, it does not necessarily lead to a reduction in the storage overhead, since nodes must still store all the scripts/signatures that are relevant for the verification of unspent transactions.

III. Dynamics of Bitcoin Storage

In this section, we analyze the Bitcoin ledger in order to understand the underlying dynamics of blocks, UTXOs and, more in general, data utilization in the blockchain. Our goal is to extract insights that may aid the design of a storage-saving strategy.

We conducted these experiments by leveraging two existing open source tools, namely `bitcoin-blockchain-parser` [5] and `bitcoin-tools` [6], both written in Python. The `bitcoin-blockchain-parser` parses Bitcoin's raw data stored on disk by Bitcoin's software (`bitcoind`). We use it to scan Bitcoin's historical data (e.g., `blk*.dat` files) for the range of blocks to be analyzed, and for the entire blockchain to apply a storage optimization method. The library `bitcoin-tools` parses Bitcoin's block index and chainstate. The block index stores information for every block (e.g., block header and number of transactions in that block), and where each block is stored on disk. The chainstate stores Bitcoin's current UTXO set. Our tool leverages `bitcoin-tools` to fetch and decode the UTXO set. We used these parser to analyze the Bitcoin blockchain from the genesis block until block 684,816 (minted on May-25 2021).

A. *UTXO Distribution and Lifespan*

We start by looking at the "current" UTXO set (i.e., UTXO set at block 684,816) made of 123,394,434 UTXO in total. As shown in [7], a very large number of UTXOs are "dormant", i.e., they have been confirmed early in the blockchain's history and not spent thus far. Namely, our measurements show that most of the blocks (84%) confirm at least one of the current UTXOs. The first 105,000 blocks (i.e., between Jan-09 2009 and Jan-28 2011) account for 42,802 of the UTXOs in the current UTXO set. By examining the latest 25,000 blocks, we only find 642 UTXOs.

There may be multiple reasons why a UTXO has been created a long time ago and never been spent. One possible reason is that the owner is holding those UTXOs speculating that BTC value will increase. Other options are, e.g., dust UTXO or UTXO belonging to secret keys that have been lost. Pérez-Solà et al., [17] studied dust or unprofitable UTXOs in Bitcoin and found that, depending on the spending fee, up to 50% UTXOs at block 507,964 (minted on Feb-6 2018) could be considered dust.

In order to distinguish whether holding UTXOs for a very long time is common practice throughout the Bitcoin blockchain lifetime, we look at the lifespan of a UTXO, that is, the time—measured in blocks—between its addition to the blockchain (i.e., its parent block is added to the chain) and its expenditure (i.e., the block storing a transaction that spends the UTXO is added to the chain) at different time snapshots in the Bitcoin blockchain. Figure 3 shows the 50th, 90th, and 95th percentile of the UTXO lifespan across different time intervals. The first interval comprises the time from the genesis block until block 104,999, minted on Jan-28 2011: the blue curve of Figure 3 shows that less than 75% of the UTXOs created within blocks 0-104,999 were actually spent before block 104,999. The second time interval goes from the genesis block until block 419,999, minted on Jul-09 2016: the corresponding dashed orange curve shows that UTXOs created within blocks 0-419,999 have a different behavior from the ones in the previous interval; roughly 90% of those UTXOs have a lifespan smaller than 100,000 blocks. Finally, the third interval consider all blocks up to 684,816 (minted on May-25 2021): here we witness (green curve with dashes/dots) a trend similar to the one in the previous interval as 90% of the UTXOs are spent within 42,000 blocks.

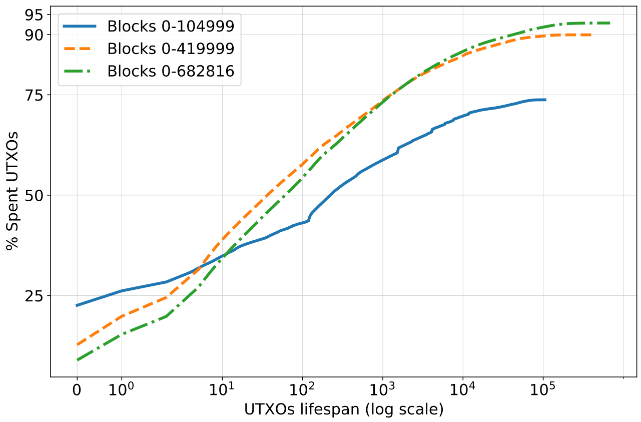

Fig. 3. Lifespan (in blocks) of UTXOs created during the first 105,000, 420,000, and 682,816 blocks.

In a nutshell, Figure 3 shows that, albeit a large number of UTXOs are dormant—especially the ones created during the early years of Bitcoin—most of the UTXOs have a rather short lifetime. More precisely, we note that 90% of all UTXOs are currently spent within 42,586 blocks (cf. Figure 3). As Bitcoin becomes increasingly popular, we expect the UTXO lifespan to decrease over time.

B. *Data Distribution within Bitcoin Transactions*

Next, we focus on the various data types that are included in a Bitcoin transaction and compute the fraction of transaction data used to store the block header, transaction header, TxIns, TxOuts, and scripts. Here, we do not distinguish between the individual scripts—rather we measure deduplication level in scripts regardless of their type. To take into account segregated witness data, we divide the analysis in two different time-periods: the period before the introduction of segwit, and the one after the introduction of segwit.

Figures 4 show the storage fraction that each data type occupy in a Bitcoin transaction, for transactions between block 0 and block 481,823, i.e., before the introduction of segwit. Figures 5 provides the same information but for transactions between block 481824 and 684816, that is, after the introduction of segwit. Our results show that—perhaps unsurprisingly—transaction scripts account for more than 50% of the size of a transaction.

For example, Figure 4 reveals that input scripts and output scripts account for about 62% and 12% of a transaction size, respectively. Figure 5 shows that segregated witnesses reduced the fraction of storage used for input scripts, while adding about 25% of witness data. The space ratio of other data types remain, as expected, unaffected by segwit.

Given the impact that scripts have on the size of a transaction, we investigate the duplication level of scripts within Bitcoin's ledger. Table I, shows that—in addition to their considerable size—there is large amount of duplicated data among Bitcoin's script. Concretely, Table I shows that about 4,000,000 scripts stored in a TxIn and 40,000,000 scripts stored in a TxOut repeat at least twice. The huge difference between TxIn and TxOut scripts duplication may be caused by unclaimed coins, as well as invalid scripts, or unredeemable transactions (e.g., OP_RETURN). Nevertheless, our results in Table I shows that in spite of their high duplication, scripts

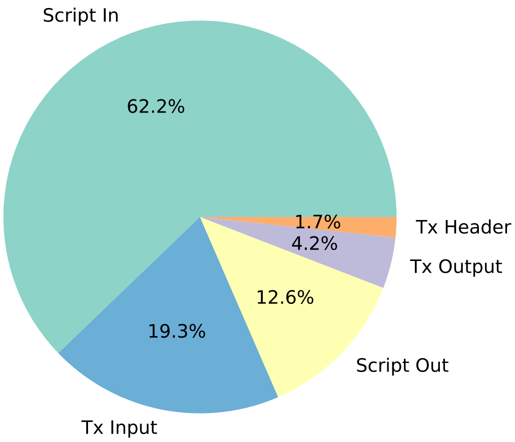

Fig. 4. Distribution of data types within transactions from block 0 to block 481,823 (i.e., prior to the introduction of segwit).

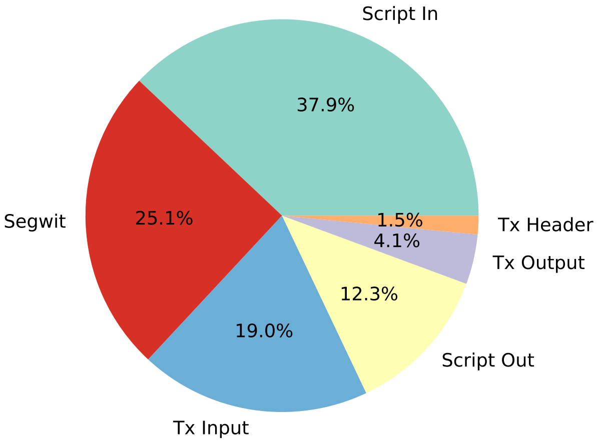

Fig. 5. Distribution of data types within transactions from block 481,824 to block 684,816 (i.e., after the appearance of segwit data).

only occupy a small fraction of the Bitcoin ledger storage (approximately 1.5 GB).

C. Unused Bytes in Bitcoin Transactions

Finally, we look at unused bytes in each of a transaction's fields. In particular, we observe that some fields in a Bitcoin transaction have a (large) fixed size; these fields have been designed with the foresight to scale as Bitcoin adoption increases. At present time, the following fields are however largely unused.

- **Transaction headers**:
  - *version*: 4 bytes encoding the transaction's data format version. Notice that there only 2 versions available at the time of writing.
  - *flag*: 2 bytes indicating the presence of witness data.
  - *lock_time*: 4 bytes encoding either the block number or the timestamp at which this transaction becomes unlocked. If the transaction does not have a time lock, this field has a default value of `0xFFFFFFFF`.
- **TxIns**:
  - *previous_output*: 36 bytes are reserved to encode the hash of the transaction referenced by this input (32 bytes) and the index of the specific output to spend in that transaction (4 bytes). Notice that it might suffice to reference the block height and the transaction index at that block height (4 bytes + 2 bytes) along with a variable length integer to encode the specific output to spend in the transaction.
  - *sequence*: 4 bytes that determine the transaction version as defined by the sender. There only 2 versions available at the time of writing.
- **TxOuts**:
  - *value*: 8 bytes to encode the value to be spent (which is a rather large field size).

IV. Local Storage Optimizations

Based on the observations provided in Section III, we explore the space of possible solutions that allow a Bitcoin node to reduce the ledger's footprint on its local storage. Since we are looking at solutions that can be applied by a node to its local storage without aid by its peers and without modifications to the underlying protocol, we discard techniques that require cooperation among nodes or changes to the underlying protocol (cf. Section II).

The Bitcoin whitepaper originally foresaw two different roles in the Bitcoin blockchain: a *full node* and a *lightweight client*. Lightweight clients cover the needs of basic blockchain users that send or receive transactions but entrust peers to validate transactions. Full nodes were originally designated to contribute to the consensus protocol, by storing and verifying all of the blockchain data, as well as mining new blocks. As Bitcoin grew in adoption and the difficulty of mining increased, a new dedicated *miner* role emerged; these nodes are not meant to store or verify blockchain data, but they are merely optimized to compute PoW solutions.

Beyond these traditional roles, a new type of node is becoming popular: the *archival* node. Such nodes store a full copy of the blockchain for offline auditing and verification purposes but are not interested in real-time verification of transactions.

Given the different types of Bitcoin nodes, one can envision different space-saving strategies that depend on the role of nodes in the ecosystem. For instance, an aggressive data-saving strategy that favors space reduction over the ability to verify all transactions may work for some node types, whereas others might value verifiability of information more than storage savings. In the following, we explore the space of storage-saving options rooted on the observations of Section III.

A. Storage Optimization Toolbox

Pruning based on UTXO lifespan (PRUNE). The pruning functionality offered by the Bitcoin client is a lossy mechanism and, in case pruned UTXOs are spent, the node must fetch data from the blockchain for verification (thus incurring additional communication overhead). Our measurements in Section III

Script Type	# of Duplicated Scripts	avg length	Tot. size	Tot. size (Dedup.)
TxIn script + segWit	$\sim 4\,100\,000$	37.6 B	177 MB	16 MB
TxOut script	$\sim 40\,500\,000$	24.8 B	1.3 GB	200 MB

TABLE I
DUPLICATION OF SCRIPTS IN THE BITCOIN BLOCKCHAIN UNTIL BLOCK 545,000.

provide solid means to choose a pruning threshold, based on the probability that one of the pruned UTXOs is spent. According to Figure 3, UTXOs that are older than 42,586 blocks can be pruned, if one tolerates that with 10% probability one of the pruned UTXO will be spent.

Given the moderate storage costs of hashes, we argue that the block hashes should be kept in this strategy—even for those blocks that are pruned. As discussed in Section IV-B, this ensures that the security of the system is not compromised against sophisticated attacks.

Minimizing Merkle-tree data (MINIMIZE). We note that transactions in a block are arranged as leaves of a Merkle tree, so that the root can be used as the authentication token. Rather than keeping all leaves of the tree, one could simply keep the "co-path" of transactions with unspent UTXO. Given a block with n transactions, the co-path to verify an unspent one, amounts to roughly $\log n$ nodes of the Merkle tree. Hence, given k unspent transactions in a block, the amount of tree nodes to be stored is $k \cdot \log n$. Given that a full binary tree has $n - 1$ internal nodes, the strategy of keeping the co-path of unspent transactions reduces storage as long as $k < n/\log n$.

Notice that approximately 16% of the blocks have no UTXOs and can therefore be removed without any penalty. For those blocks, no intermediate information about the Merkle root/tree must be stored. Nodes can simply store the block hash as a means to compute the longest chain.

Slack space reduction (SLACK). As shown in Section III, there is considerable room to remove the slack space in several transaction header fields, namely:

- *version*: Since there are currently only 2 transaction versions, we can limit the size of this field to 1 bit instead of 4 bytes.
- *flag*: Given that this field is only used to indicate the presence of witness data, we can also limit its size to 1 bit.
- *lock_time*: One can adjust this field as follows: if the transaction does not have a time lock, we include a bit flag. Otherwise, we leave the *lock_field* set time.

Similar techniques can also be applied to the transaction input fields, namely:

- *previous_output*: One can adjust this field to use a combination of block height and referenced transaction index at that block height (4 bytes + 2 bytes) instead of the 32 bytes hash of the transaction, and a variable length integer to encode the specific output to spend in the transaction.
- *sequence*: Since there are currently only 2 transaction versions, we can limit the size of this field to 1 bit instead of 4 bytes.

Moreover, one can also optimize slack space in the transaction outputs by modifying the current 8-byte *value* field to accommodate for a variable length integer field.

Finally, one can, in theory, deduplicate existing scripts by implementing a key-value store for efficient script storage. In particular, given a duplicated script, one can store it in the key-value store (KVS), indexed by its hash, and later replace the script with its hash in every transaction when the script appears. The local node would then fetch scripts from the key-value store, any time the transaction needs to be verified. However, our experiments show that such a strategy would result in considerable I/O (to access the KVS) only to result in modest storage savings. Namely, as shown in Table I, scripts (that could be deduplicated) occupy approximately 1.5 GB of storage; replacing such scripts with KVS pointers would only yield a modest saving of 1.2 GB (0.3% of the total Bitcoin ledger storage).

B. *Storage-efficient modes*

We now analyze the security and performance tradeoffs of the various storage-saving strategies discussed above. In particular, for each strategy, we analyze the storage savings and the security implications for a node adopting a particular strategy, in comparison with a full node—one that stores the full blockchain. In terms of security, we focus on whether a node can validate all transactions or trace back a coin expenditure throughout the entire ledger.

In our analysis, we distinguish between "lossy" and "lossless" strategies. A lossless strategy, as the name suggests, does not imply any information loss compared to the option of storing the full blockchain but might imply a computational penalty (see discussion below) when verifying transactions. Differently, a lossy strategy incurs in loss of information when compared to storing the full blockchain and might incur a communication penalty when verifying transactions—as missing information must be fetched from the network.

Notice that SLACK is a lossless strategy and, as such, have no impact on security. That is, a node using SLACK preserve the same ability of a full node of verifying transactions and tracing back coins.

On the other hand, MINIMIZE and PRUNE are lossy strategies. PRUNE does not allow to verify transactions that include those UTXOs that have been removed. Moreover, both strategies do not allow nodes to trace coin expenditure, since some information (e.g., transactions confirmed in blocks prior to the pruning threshold in PRUNE or transactions for which the UTXOs are spent in MINIMIZE) has been removed. Hence, a node that uses MINIMIZE and/or PRUNE may need to contact a full or archival node to obtain the missing

TABLE II
EVALUATION OF THE VARIOUS STORAGE-SAVING STRATEGIES.

Storage Strategy		Storage overhead (GB)	Storage reduction (%)
Full-ledger storage (baseline)		371.4 GB	0%
Standard Compression	snappy	335.4 GB	9.7%
	lzop	325.3 GB	12.41%
	lz4	318.1 GB	14.35%
	bzip2	302.8 GB	18.47 %
	gzip	300.9 GB	18.98%
	zstd	294.9 GB	20.59%
	lzma	279.6 GB	24.71%
Storage-efficient Modes	PRUNE	51.20 GB	86.22%
	SLACK	265.1 GB	28.62%
	MINIMIZE	54.4 GB	85.3%
	MINIMIZE + PRUNE	16.5 GB	95.56%
	MINIMIZE + SLACK	50.6 GB	86.37%
	PRUNE + SLACK	42.4 GB	88.58%
	PRUNE + MINIMIZE + SLACK	15.2 GB	95.90%

data when a transactions with a pruned UTXO must be verified or a coin must be traced back. Nevertheless, we argue that MINIMIZE and/or PRUNE do not affect the ability of a node to verify transactions including UTXOs that have not been deleted. In particular, even if not all UTXOs are kept, a node that uses either strategy still keeps the block hashes. Thus, a fraudulent transaction would be considered as valid only if the adversary is able to find a hash collision. In case of PRUNE, a proper choice of the pruning threshold is particularly important. One needs to set it high enough so that the vast majority of UTXOs can be verified locally (without the need to fetch data from other peers) but low enough not to occupy large storage space.

Notice that these strategies can be combined with each other to increase the utility of the nodes—given the role they envision to take in a blockchain (i.e., active verifier, or passive archival node). In particular, archival nodes can easily adopt SLACK, while active verifiers could use a combination PRUNE and MINIMIZE, optionally with SLACK.

V. EVALUATION

In this section, we evaluate the effectiveness of the strategies discussed in Section IV. To do so, we created a software tool in Python that can be executed locally to estimate the storage footprint of the Bitcoin ledger, given any combination of the storage-saving strategies we devise. The tool takes as input any combination of the storage-saving strategies of the previous section. Subsequently, it parses the Bitcoin ledger and outputs the corresponding storage footprint needed to store the ledger. As mentioned earlier, our tool adapts two existing open-source parsers, namely `bitcoin-blockchain-parser` [5] and `bitcoin-tools` [6], both written in Python.

We conducted our experiments on a machine equipped with an Intel®Xeon®CPU E-2176G @ 3.70GHz and 128GB DDR4 RAM.

Storage savings: We compare the performance of our space-saving strategies against two different baselines: the full current ledger, and a ledger compressed with standard compression algorithms, namely:

- `bzip2`: lossless compression using the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding.
- `gzip`: lossless compression using Lempel-Ziv coding (LZ77).
- `lzma`: lossless compression using a dictionary compression scheme similar to Lempel-Ziv coding (LZ77).
- `lzop`: lossless compression using the Lempel–Ziv–Oberhumer (LZO) algorithm.
- `lz4`: lossless data compression algorithm focused on compression and decompression speed. Based on Lempel-Ziv coding (LZ77).
- `snappy`: lossless compression algorithm developed by Google. It does not aim for maximum compression, or compatibility with any other compression library; instead, it aims for very high speeds and reasonable compression [2].
- `zstd`: lossless compression algorithm developed by Facebook. It is a fast lossless compression algorithm, targeting real-time compression and better compression ratios [1].

To apply standard compression to the Bitcoin ledger, we first created a single non-compressed archive file of the ledger using `tar`, then fed it to the compression algorithm.

We note that compressing the ledger typically entails trading-off data saving for performance. In particular, one could compress the whole ledger to obtain the best savings in terms of space; however, verifying a transaction would require de-compressing the ledger and that may incur considerable delays. Alternatively, one could compress one block at a time: this is likely to save less space but, at the same time, it is likely to perform better when verifying a transaction since one has to de-compress only some blocks. Our measurements, however, show that even the best compression strategy (i.e., compressing the whole ledger) only yields modest results. For instance, among all studied compression algorithms, `lzma` achieves the best storage saving and only results in around a 24% storage saving. When compared to SLACK, the latter strategy achieves a 5% increase (i.e., 29%) in storage savings (cf. Table II) while ensuring zero information loss from the Bitcoin blockchain; it also exhibits significant reductions in computational load required for compression/de-compression.

As shown in Table II, other strategies outlined in Section IV achieve much higher storage efficiency—while requiring less computational overhead. For instance, MINIMIZE results in 94% storage savings and PRUNE can achieve a savings of 86% while ensuring that 90% of all UTXOs can be verified with the local data storage (cf. Section III).

Various combinations of the aforementioned strategies seem to be also very effective. For instance, the combination of SLACK and PRUNE results in 88.58% storage savings, while a straightforward combination of MINIMIZE and PRUNE results in up to 95.56% storage savings. Finally, the combination of PRUNE, MINIMIZE, and SLACK can result in huge storage savings, up to 95.90%. That is, the reliance on SLACK would only result in mediocre storage gains when PRUNE and MINIMIZE are being used.

Computational overhead: In order to assess the computational overhead of the proposed strategies, we additionally measured the time it took to our tool to run MINIMIZE and SLACK over each block between block 682,807 and block 682,816 of the Bitcoin blockchain. On average. MINIMIZE took 2.1ms (std-dev 0.9ms) whereas SLACK took 102.9ms (std-dev 58.7ms). We conclude that both strategies incur in negligible processing overhead. We did not evaluate the overhead incurred in PRUNE since it merely requires the peer to *delete* a given block.

VI. Concluding Remarks

In this paper, we tackled an often overlooked issue of today's blockchains—reducing the ledger size.

We observe that current blockchains do not employ storage-efficient strategies, their ledger features considerable redundancy, and they rarely utilize all the data stored in the ledger for daily operations. We apply our reasoning to Bitcoin's data storage, and show, by means of empirical measurements, that the ledger storage can be considerably reduced without modifying the underlying consensus protocol nor affecting the security of the verification process. To this end, we adapted the blockchain parser from [5] and measured the storage footprint of various local strategies that can be directly employed at client-side by full nodes in Bitcoin.

Our evaluation results show that standard compression algorithms are not effective in capturing the intrinsic nature of Bitcoin's ledger and only result in up to 24% storage savings. On the other hand, more fine-grained lossless compression strategies—those that specifically target unused bytes or duplication in the blockchain—prove to be more effective and could lead to storage savings up to 29%.

In terms of lossy strategies, our results also show that ledger pruning—a popular strategy to reduce the local storage footprint of nodes—would require at least 51 GB of storage space in Bitcoin in order to process the vast majority of those UTXOs in circulation. Besides laying the grounds that govern the effective choice of a suitable pruning threshold, we also show that pruning can be combined with other lightweight strategies to incur a lower storage footprint, as low as 15.2 GB without incurring significant computational overhead on nodes.

We plan to release our parser as open-source to better aid the community in estimating the actual storage needs of Bitcoin nodes as the ledger grows in size. Finally, we stress that our observations and results are not restricted to the Bitcoin blockchain and equally apply to the myriad of altcoins (or forks of the Bitcoin blockchain) that are currently deployed (e.g., Dogecoin, Bitcoin Cash, Litecoin, Monacoin).

References

[1] GitHub - facebook/zstd: Zstandard - Fast real-time compression algorithm. https://github.com/facebook/zstd.

[2] GitHub - google/snappy: A fast compressor/decompressor. https://github.com/google/snappy.

[3] . https://github.com/bitcoin/bitcoin/blob/v0.11.0/doc/release-notes.md#block-file-pruning, 2015.

[4] . https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki, 2015.

[5] GitHub - alecalve/python-bitcoin-blockchain-parser: A Python 3 Bitcoin blockchain parser. https://github.com/alecalve/python-bitcoin-blockchain-parser, 2015.

[6] GitHub - sr-gi/bitcoin_tools: Python Bitcoin tools. https://github.com/sr-gi/bitcoin_tools, 2016.

[7] Bitcoin dormant coins . https://stats.buybitcoinworldwide.com/dormant-coins/, 2021.

[8] Cambridge Bitcoin Electricity Consumption Index (CBECI). https://cbeci.org/index, 2021.

[9] J. Bruce. The mini-blockchain scheme. *White paper*, 2014.

[10] A. Chepurnoy, M. Larangeira, and A. Ojiganov. Rollerchain, a blockchain with safely pruneable full blocks. *arXiv preprint arXiv:1603.07926*, 2016.

[11] M. Florian, S. Henningsen, S. Beaucamp, and B. Scheuermann. Erasing data from blockchain nodes. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 367–376. IEEE, 2019.

[12] The Decline in Bitcoin Full Nodes, 2015. Available from https://bravenewcoin.com/news/the-decline-in-bitcoins-full-nodes/.

[13] A. Marsalek, T. Zefferer, E. Fasllija, and D. Ziegler. Tackling data inefficiency: Compressing the bitcoin blockchain. In *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)*, pages 626–633, 2019.

[14] R. Matzutt, B. Kalde, J. Pennekamp, A. Drichel, M. Henze, and K. Wehrle. How to securely prune bitcoin's blockchain. In *2020 IFIP Networking Conference*, pages 298–306, 2020.

[15] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008.

[16] E. Palm, O. Schelén, and U. Bodin. Selective blockchain transaction pruning and state derivability. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 31–40. IEEE, 2018.

[17] C. Pérez-Solà, S. Delgado-Segura, G. Navarro-Arribas, and J. Herrera-Joancomartí. Another coin bites the dust: An analysis of dust in UTXO based cryptocurrencies. *IACR Cryptol. ePrint Arch.*, pages 1–26, 2018.