computers

MDPI

Article

Unlocking Blockchain UTXO Transactional Patterns and Their Effect on Storage and Throughput Trade-Offs

David Melo [1], Saúl Eduardo Pomares-Hernández [1], Lil María Xibai Rodríguez-Henríquez [1,2,*] and Julio César Pérez-Sansalvador [1,2]

[1] Instituto Nacional de Astrofísica, Óptica y Electrónica Santa María Tonantzintla, Puebla 72840, Mexico; jdom1824@inaoe.mx (D.M.); spomares@inaoep.mx (S.E.P.-H.); jcp.sansalvador@inaoep.mx (J.C.P.-S.)

[2] Investigadoras e Investigadores por México, CONAHCYT Av. Insurgentes Sur 1582, Col. Crédito Constructor, Del. Benito Juárez, Mexico City 03940, Mexico

* Correspondence: lmrodriguez@inaoep.mx

Abstract: Blockchain technology ensures record-keeping by redundantly storing and verifying transactions on a distributed network of nodes. Permissionless blockchains have pushed the development of decentralized applications (DApps) characterized by distributed business logic, resilience to centralized failures, and data immutability. However, storage scalability without sacrificing throughput is one of the remaining open challenges in permissionless blockchains. Enhancing throughput often compromises storage, as seen in projects such as Elastico, OmniLedger, and RapidChain. On the other hand, solutions seeking to save storage, such as CUB, Jidar, SASLedger, and SE-Chain, reduce the transactional throughput. To our knowledge, no analysis has been performed that relates storage growth to transactional throughput. In this article, we delve into the execution of the Bitcoin and Ethereum transactional models, unlocking patterns that represent any transaction on the blockchain. We reveal the trade-off between transactional throughput and storage. To achieve this, we introduce the spent-by relation, a new abstraction of the UTXO model that utilizes a directed acyclic graph (DAG) to reveal the patterns and allows for a graph with granular information. We then analyze the transactional patterns to identify the most storage-intensive ones and those that offer greater flexibility in the throughput/storage trade-off. Finally, we present an analytical study showing that the UTXO model is more storage-intensive than the account model but scales better in transactional throughput.

Keywords: blockchain scalability; permissionless blockchain; decentralized applications; UTXO model; account model

Citation: Melo, D.; Pomares-Hernández, S.E.; Rodríguez-Henríquez, L.M.X.; Pérez-Sansalvador, J.C. Unlocking Blockchain UTXO Transactional Patterns and Their Effect on Storage and Throughput Trade-Offs. *Computers* **2024**, *13*, 146. https://doi.org/10.3390/computers13060146

Academic Editors: Caterina Tricase, Otar Zumburidze, Nino Adamashvili, Radu State and Roberto Tonelli

Received: 30 April 2024
Revised: 28 May 2024
Accepted: 4 June 2024
Published: 7 June 2024

1. Introduction

Blockchain technology is an innovative digital ledger system that provides secure record-keeping by storing and redundantly verifying transactions on a distributed network of nodes [1]. This technology bifurcates into two primary classes: public (or permissionless) and private (or permissioned) blockchains. Permissionless blockchains are open access and allow the participation of any individual or entity [2], while permissioned blockchains require credential validation or an economic incentive to allow collaboration in the network [3]. Permissionless blockchains have pushed the development of DApps, which exhibit features such as distributed business logic, distributed data, resilience to failures at central points, and a guarantee of data immutability [4].

However, permissionless blockchains face challenges that limit the optimal operation of DApps. One of the most relevant challenges is storage scalability, specifically the growth of the blockchain's sublinearly with the number of nodes. To understand the problem of storage scalability in blockchains, let us imagine a library that constantly receives new books (blockchain transactions) with a constant daily rate of ten books, known as the growth rate, c. For security and redundancy, the library stores copies of all the received books in different sections, with the number of sections equivalent to the number of nodes n. In this

scenario, if we want to determine the total number of books in the library storage size, s, we could calculate it as $s = c \times n$. However, the challenge occurs when the librarian cannot control the number of sections (nodes) where the book copies are stored. For example, one day there are five sections, and the next day, there are seven sections. This fluctuation in the number of sections affects the storage capacity of the library and the management of the books.

A real-world example of this challenge is seen in Bitcoin, where the storage size of the blockchain has currently reached 3.28 petabytes [5]. This situation is influenced by the constant growth rate of the blockchain, which is approximately 488 GB per node, and by the number of nodes redundantly storing transactions, presently around 7065 [5]. Ethereum [6] serves as a notable case where storage growth may follow an exponential trend, as depicted in Figure 1.

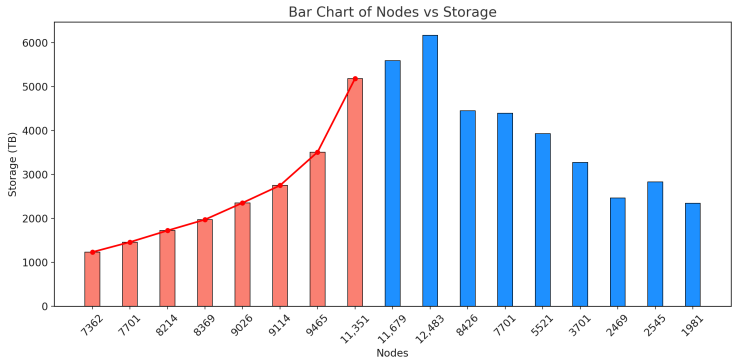

Figure 1. Growth trend of Ethereum storage capacity. The bar chart illustrates the exponential growth in Ethereum's storage demand over time, peaking at 12,483 nodes and requiring nearly 6000 terabytes of storage.

The previously mentioned issues arise from the inherent redundancy built into the design of permissionless blockchains. This redundancy creates a delicate balance: improvements in transactional throughput (measured in transactions per second) inevitably lead to increased storage requirements, while attempts to reduce storage potentially compromise throughput due to decreased availability and increased latency.

There are three primary approaches to increasing transactional throughput: block size management, off-chain mechanisms, and sharding. Block size management increases the block size to allow more transactions per block, temporarily helping transaction congestion [7,8]. Off-chain mechanisms process transactions outside the main blockchain through payment channels or sidechains, reducing the load on the main blockchain [9–12]. Sharding increases throughput by splitting the blockchain into smaller, parallel-processing parts called shards [13–15]. However, the impact of these approaches on storage growth needs careful consideration.

Storage efficiency enhancement approaches are divided into centralized and decentralized data. Centralized approaches store data in a single location or through a central entity [16–18], while decentralized strategies distribute data across multiple nodes in the blockchain network, enhancing robustness and immutability [19,20]. The common goal is to increase storage efficiency, but these strategies affect transactional throughput.

In summary, advances in blockchain technology aim to enhance transactional throughput and reduce node storage requirements. However, these goals are not mutually exclusive, as improvements in one often impact the other. We identified a noticeable gap in the analyses that relates storage growth to transactional throughput and vice versa. In this article, we unlock transactional patterns of the UTXO model to reveal the relation between storage and transactional throughput, providing the first analysis of the relation of these parameters. To achieve this, we apply the following methodology:

1. Analysis and abstraction of transactional models.

2. Formal comparison of models to highlight their cost on storage.
3. Run experiments with data from the Bitcoin and the Ethereum blockchains.

The analysis resulting from the previous methodology shows that the UTXO model is more storage-intensive but offers flexibility in transactional throughput, showing signs of a trade-off in the parameters. The transactional behavior of the models, resulting from the abstraction step, led us to introduce a novel DAG-based abstraction of the Bitcoin transactional model: the spent-by relation. This new relation unlocks the transactional patterns that represent any transaction on the blockchain and shows the relationship between throughput and storage. Finally, the experiments on more than 800 M transactions show the most storage-intensive transactional patterns.

The remainder of the paper is structured as follows: Section 2 presents an overview of the fundamental concepts of transactional models. Section 3 presents an overview of related work, with particular emphasis on strategies that impact storage/throughput within blockchain systems. Section 4 presents an analysis of the execution of transactional models and their impact on blockchain storage. In Section 5, the spent-by relation is introduced as a novel abstraction of the UTXO model. In Section 5.3, we unlock the transactional patterns within the UTXO model. Finally, in Section 6, we introduce an experimental comparison of storage costs in UTXO transactional patterns.

2. Fundamental Background of Transactional Models

Blockchain technology, at its most basic essence, provides a mechanism for secure and verifiable storage of records through a redundancy system. This redundancy results from the verification and distributed storage of transactions in a network of nodes operating in a peer-to-peer (P2P) system. Transaction records on the blockchain network are grouped into blocks, thus creating a chain of blocks, hence the term "blockchain". Each block contains a series of transactions, all of which are validated and confirmed by the network. The block is linked to the previous one through a unique identifier called hash. This hash results from a cryptographic function that takes the data of the current block and the ID from the previous one, producing a unique fixed-length string. This implies that any change breaking the blockchain indicates manipulation.

This fundamental understanding of blockchain technology sets the stage for a deeper exploration of its complexity and functionality, especially in the context of the transactional models of Bitcoin and Ethereum. In this section, the main transactional models are discussed, specifically the unspent transaction output (UTXO) [21] model and the account model [22].

2.1. UTXO Model

In the UTXO model, the state of transactions is represented as a collection of unspent transaction outputs. This is illustrated in the DAG shown in Figure 2, where vertices symbolize transactions, and edges represent pointers that consume the previous transaction to generate a new one.

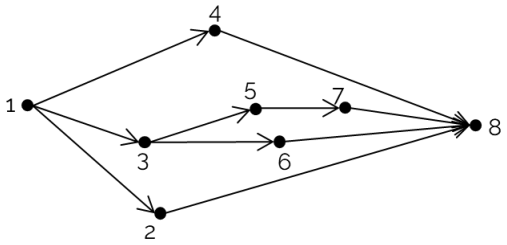

Figure 2. Graphical representation of a DAG showing the flow of transactions in the UTXO model from a Coinbase output (1) to a single input (8), noting the divergence and convergence of paths.

There are several definitions of the UTXO model, such as a Directed Acyclic graph. In this article, the definition provided by Jeyakumar et al. [23] is highlighted for its ability to encompass the transactional model of Bitcoin and Ethereum.

Definition 1. *A directed graph $G(V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ represents the set of nodes and $E \subseteq V \times V$ represents the set of edges. For each vertex v_i, an edge e_i is of the form $v_i \rightarrow v_j$.*

Bitcoin transactions use one or more unspent outputs from previous transactions to create new outputs. These new outputs become unspent outputs that are available for future transactions.

According to Narula and Dryja [24], a digital signature, a public key, and a timestamp must be provided to consume an unspent output. In addition, the following properties must be met:

1. All outputs are not the same.
2. An unspent output refers to a specific output when spending.
3. Unspent outputs are consumed, creating new outputs.
4. An output can only be spent once.

These properties are based on the Bitcoin protocol and replicated in other applications.

2.2. Account Model

In contrast to the UTXO model, the account model represents the state of blockchain transactions as a variety of accounts or addresses, which are managed by entities or smart contracts [25]. These entities can be individuals, organizations, or automated systems. An example of automated systems is smart contracts, which are simple programs housed within Ethereum's virtual machine (EVM), facilitating the execution of complex operations and agreements autonomously, while providing high reliability.

In Ethereum's implementation of the account model, transactions are abstractly represented as state transitions. Figure 3 shows a graphical representation that illustrates the flow of transactions that update account statuses as they are executed.

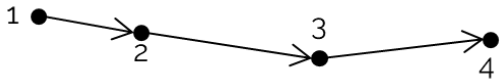

Figure 3. Serialized graph, illustrating the transaction sequence in the account model from the origin node (1) to the end node (4).

In addition to the traditional transactions in Ethereum, there are types of transactions specifically related to smart contracts. These transactions are typically classified in the literature as contract deployment and contract invocation:

- The process of contract deployment essentially involves the creation of a smart contract. This can be equated to an executable program that is assigned a unique address within the blockchain. The smart contract contains a set of predefined functions or instructions that are written in a programming language compatible with the Ethereum blockchain, such as Solidity [26].
- On the other hand, contract invocation refers to the process of executing or "calling" the functions embedded within the smart contract. These functions can be invoked by other addresses within the blockchain network, allowing them to interact with the smart contract and initiate specific operations. These operations can range from simple value transfers to more complex interactions involving multiple smart contracts [27].

Finally, there are other transactional models, such as the EUTXO model [28] and the account abstraction ledger [29], but these are based on the models discussed above.

3. Related Work

As previously discussed in the introduction, strategies to save storage or increase transactional throughput have been addressed in a disjointed manner.

This section analyzes storage improvement within two strategies: centralized and decentralized. At the same time, we examine approaches designed to improve throughput based on sharding, off-chain, and block size. Figure 4 categorizes each method based on improvements in throughput and storage parameters. In particular, strategies that reduce storage tend to reduce transactional throughput, as depicted in the top left. Contrarily, methods that increase transactional throughput tend to be storage-intensive, as shown in the lower right. After classifying each method, we found a noticeable gap in the literature: a lack of studies investigating the trade-off between throughput and storage based on blockchain transactional models.

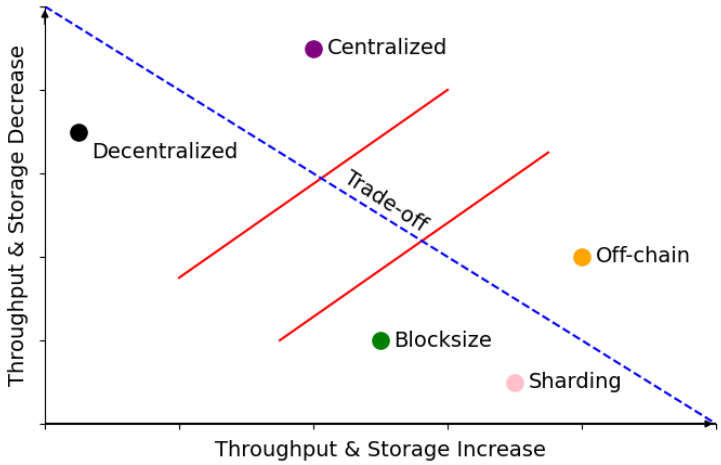

Figure 4. Scatter plot showing the dilemma faced by blockchain environments in the parameters of transactional throughput and storage efficiency. The dots indicate proposals to improve one of the two parameters, including decentralization, centralization, block size, off-chain strategies, and sharding.

3.1. Approaches to Enhance Throughput

Strategies to enhance transactional throughput are primarily focused on increasing the number of processed transactions within a given time frame. The approaches are categorized in Table 1, organized in ascending order based on the level of transactional throughput they achieve. We describe the advantages and disadvantages associated with each method as follows.

3.1.1. Block Size

The first approach to enhance transactional throughput involves increasing the block size. This strategy has been used in cryptocurrencies, such as Bitcoin and Ethereum, where the block size has been increased to allow for the inclusion of more transactions, thereby temporarily alleviating transaction congestion [7,8,30]. However, this design decision has disadvantages since larger blocks require more storage resources and take longer to process and propagate over the network.

3.1.2. Off-Chain

The off-chain transaction approach mainly uses two methods: (a) payment channels and (b) sidechains, which effectively enhance transactional throughput while simultaneously impacting storage requirements on the main blockchain.

(a) Payment channels increase transactional throughput by creating private paths between entities. For example, the Lightning Network [9,10] is capable of handling up to one million TPS off the main blockchain, recording transactions on the blockchain

only when the channels are closed. This method shifts the storage overhead from the main chain to external systems to increase transaction throughput, altering the balance between these two parameters. In addition, managing the states of the channels off the main chain requires additional resources to ensure the integrity of the transactions.

(b) Sidechains [11,12] operate as independent blockchains with their own storage and consensus mechanisms, linked to the main chain by two-way pegs. This setup allows them to process transactions that do not burden the main chain, enhancing overall system performance. However, the need for additional infrastructure to maintain the security and operability of sidechains increases off-chain storage and management overhead.

Finally, regardless of the method used, the off-chain transaction approach does not offer transparency at the same level as on-chain transactions. This is because the channels are not visible to all participants, and the sidechains do not maintain the same security.

Table 1. Blockchain storage scaling-related work.

Issue	Enhancing Throughput	Proposal	Blockchain System	Throughput	Advantages	Disadvantages
Throughput	Block size	SegWit 2015	Permissionless	20 TPS	Enhances storage efficiency and latency	Increases capacity, not scalable solution.
	Block size	London 2021	Permissionless	85 TPS	Increase throughput reduce fees	Increased gas and block size
	Off-chain	Polygon 2018	Permissionless	65,000 TPS	Ethereum compatibility low fees	Inconsistency and security risks
	Off-chain	Lightning 2015	Permissionless	1 Million TPS	Instant transactions low fees	Funds blocked in payment channels
	Sharding	Elastico 2016	Permissionless	40 TPS	Parallelizes transactional processing	Increase storage 1 GB per day
	Sharding	OmniLedger 2018	Permissionless	4000 TPS	Ensures that nodes redistributed	Increase storage 28 GB per day
	Sharding	RapidChain 2018	Permissionless	7380 TPS	Efficiency in network configuration	Increase storage 159.6GB per day
Issue	**Reducing Storage**	**Proposal**	**Blockchain System**	**Saving Storage**	**Advantages**	**Disadvantages**
Storage	Centralized	CUB 2018	Permissioned	90% Saving	Block Allocation Optimization	Assume all nodes are honest
	Centralized	Jidar 2019	Permissionless	98% Saving	Only stores transaction relevant	Extra storage for transaction
	Centralized	SASLedger 2021	Permissioned	93% Saving	Guarantee integrity of the database	Requires remote database server
	Decentralized	SE-Chain 2021	Permissionless	70% Saving	The consistent blocks stored fewer replicas	High Latency Low availability
	Decentralized	Lightweight 2021	Permissioned	46% Saving	Optimization scheme based on Reed-Solomon	Limited full-scale Compatibility

3.1.3. Sharding

Introduced in research such as Elastico [13], OmniLedger [14], and RapidChain [15], is recognized as a strategy for parallelizing transactional throughput and sharing storage. This enhances the transaction processing rate and mitigates short-term storage pressures in traditional blockchains. However, storage is not sustainable in the long term. For example, in its experiments with 4000 nodes distributed across 16 shards, RapidChain processed 7380 transactions per second (TPS). Assuming an average transaction size of 256 bytes, each shard stores around 9.93 GB per day, for a total daily storage of 159.6 GB across all shards. After 60 days, the storage required per shard escalates to 600 GB, for a total of 9600 GB across all shards. This exponential growth in storage highlights the lack of a trade-off between transactional throughput and storage in these approaches.

3.2. Approaches to Reduce Storage

Strategies in this category are classified into two distinct approaches: centralized and decentralized, as illustrated in Table 1. In the case of centralized strategies, data are stored

in a single location or managed by a central entity, resulting in significant storage savings compared to decentralized storage. Conversely, decentralized strategies distribute data across multiple nodes within the blockchain network, enhancing security and availability by eliminating dependence on a single node. We delineate the advantages and disadvantages of each method and highlight the percentage of storage savings as follows.

3.2.1. Centralized Data

CUB (consensus unit-based) is a centralized proposal to solve the storage problem in industrial blockchains. ZihuanYu et al. [16] organize different subsets of nodes called consensus units that work in parallel and are based on the assumption that all nodes in the same unit must trust each other. Then, each CUB node stores only a part of the blockchain data, and the entire subset stores a full copy, reducing the storage of the nodes by 90%. However, the assumption of inherent trust among nodes is rather idealistic, especially when services such as immutability and availability need to be guaranteed. In permissionless blockchain environments, such proposals are not applicable due to the specific requirements that decentralized applications develop in these environments.

Xiaohai Dai et al. [17] proposed Jidar as a better CUB. Each node in Jidar only stores the transactions it considers relevant for processing, and stores the identifier of the other transactions in a Merkle root, reducing storage. For the synchronization of the new nodes, Jidar adds a mechanism that joins all the fragments stored in the different nodes, similar to joining the pieces of a puzzle. Jidar results show that they reduce storage by 98% compared to CUB. However, Jidar requires additional processing to generate the proof in each transaction. The availability of the blockchain is very low because a node can be offline for a long time, and it affects the synchronization of new nodes. Also, implementing this solution on a high-speed multi-chain is infeasible due to the high latency required to create new transactions.

Haolin Sun et al. [18] propose SASLedger, a centralized off-chain proposal, which relieves the storage burden of the nodes that replicate the blockchain since they use a centralized server to store the blockchain. Similarly to CUB, the nodes are divided into subsets, and each subset has a centralized server outside the system, achieving a 93% reduction in storage. The nodes that interact within the system guarantee the integrity of the database by keeping the hashes of the blocks. However, the solution is against decentralized applications, as it has a central point of failure that affects data availability.

3.2.2. Decentralized Data

SE-Chain is a protocol proposed by Da-Yu Jia et al. [19], where the system consistency affects the redundant storage. Each node in the SE-Chain works as a Bitcoin node and redundantly stores a complete copy of the blockchain. But the consistent blocks are stored in fewer replicas, i.e., the greater the depth of the block in the chain, the fewer the nodes that store the block. This strategy is inspired by decentralized file systems, such as IPFS [31] or Swarm [32]. However, reducing blockchain replicas drastically reduces availability, and a DApp, being an application that does not need third parties, is at risk of losing essential data to guarantee traceability. In addition, the search for transactions that are at a high depth would have a longer query delay.

Lightweight blockchain is a protocol proposed by Chunlin Li et al. [20], an optimization scheme based on the Reed–Solomon (RS) erasure code [33] to reduce storage overhead while ensuring the availability and reachability of the blockchain. The storage scheme is focused on resource-constrained devices, making it more accessible for IoT scenarios. Moreover, the use of RS erasure coding allows for a reduction in storage without compromising data loss in the blockchain. However, it does not specify how transactional throughput varies depending on the specific IoT scenario. Erasure coding is a complex scheme that could potentially impact throughput parameters. The effectiveness of the proposal in reducing storage costs needs to be evaluated in permissionless blockchains to verify its benefits.

3.3. Summary

Finally, this section identifies a gap in the current research landscape: there is a lack of studies regarding the relation between throughput and storage in permissionless blockchain. This gap is evident in Table 1, where it is clear that existing research focuses on transactional throughput or storage efficiency, but not both. We have identified that the relation between transactional throughput and storage is complex. Understanding the variables in this relation must be approached from the perspective of transactional models. Therefore, this paper aims to fill a research gap by suggesting that understanding the relationship between storage and transactional throughput is achieved by proposing the transactional patterns in the UTXO model.

4. Understanding the Execution of Transaction Models and Their Relation to Blockchain Storage

This section analyzes the most relevant transactional models in the literature, such as the UTXO and the account model. The goal is to understand their transactional behavior and the relationship with storage. This was done by abstracting the transactional models of Bitcoin and Ethereum into transactional cases: three cases for the UTXO model and one for the account model. Using these abstractions, we performed a formal and experimental comparison and identified which of the two models incurs higher storage costs.

4.1. UTXO Model Storage Growth Analysis

In the UTXO transactional model, each transaction consumes one or more unspent outputs and generates one or more new outputs. When a new transaction is generated, it is possible to choose which unspent outputs are involved. This selection is arbitrary as long as the sum of the inputs is greater or equal to the total value of the outputs. The arbitrariness of the UTXO model allows for simultaneous operations while ensuring that the new transaction is directly linked to previous transactions on the blockchain. To better understand transaction execution consider the following example.

Example: Suppose that Alice purchases a coffee from Bob using Bitcoin. Alice has BTC 0.2as unspent outputs in her wallet, and the coffee value is BTC 0.1. Three cases can be produced after the purchase regarding how unspent outputs can be selected: (a) a single output, (b) multiple outputs with a value less than the input value, or (c) multiple outputs with the same value as the input value.

(a) In the first case, as shown in Figure 5a, Alice has a single output in her wallet with a value of BTC 0.2. To pay for the coffee, she creates a transaction that splits the BTC 0.2 unspent output into two new outputs: one with BTC 0.1 that she sends to Bob and another with BTC 0.1 that she sends back to herself.

(b) In the second case, as shown in Figure 5b, Alice has multiple outputs in her wallet with a value less than the input value. To pay for the coffee, Alice merges the unspent outputs with a lesser value up to BTC 0.1, and creates a transaction that she pays to Bob.

(c) In the third case, as shown in Figure 5c, Alice has multiple outputs in her wallet with a value equal to the input value. To pay for the coffee, Alice transfers the unspent output with the same value as the coffee and creates a new transaction that is sent to Bob.

The example above shows that the execution of the UTXO model has two features: the order selection of unspent outputs and the concurrently executed transaction. The arbitrary order of unspent output selection allows granular control over the input consumed by each transaction, allowing flexibility since a single transaction can consume multiple combinations of unspent outputs. This flexibility of the UTXO model allows for the simultaneous execution of unspent outputs. This approach facilitates the processing of multiple operations from a single unspent output within a single transaction, increasing transactional throughput. However, we have observed that this simultaneous execution in the UTXO model incurs a high storage cost. This cost escalates with an increase in the

number of new unspent outputs. This additional storage demand impacts the efficiency of these nodes' storage capabilities. A detailed analysis of the storage costs associated with the UTXO model is provided in Section 5.3.

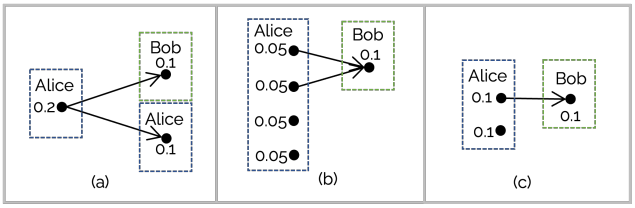

Figure 5. Transaction scenarios in the UTXO model: (**a**) Alice splits a single output of BTC 0.2 to pay Bob BTC 0.1 and returns BTC 0.1 to herself; (**b**) Alice consolidates several smaller outputs, summing up to BTC 0.1 for Bob's payment, and (**c**) Alice directly transfers an output of BTC 0.1 to pay Bob the exact amount due for the coffee, illustrating the flexibility in transaction structuring within the UTXO model.

4.2. Account Model Storage Growth Analysis

In the account model, each user has a unique address used as an identifier and associated with the balance of the transaction history. Figure 6 shows how an address's balance, as a state, is updated by transactions, which subtract transferred value assets from the sender's account and add value to the recipient's account. An example of the account model is traditional banking systems, where a user has a unique account number associated with their balance. When a user initiates a transaction, the funds are debited from their account and credited to the recipient's account. The account balance represents the current state of the user's funds, and all transactions are recorded in a ledger.

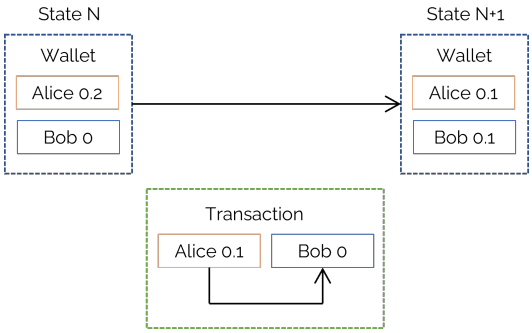

Figure 6. Illustration of the account model: The transition from State N to State N + 1 via a transaction where Alice sends 0.1 Ether to Bob, updating both their wallet balances.

Ethereum's programmability allows for two additional types of transactions within its account model: those that deploy contracts on the Ethereum virtual machine (EVM) and those invoking functions of these smart contracts. Each contract within the EVM operates under its unique set of rules and transactions, executed by external transactions. However, maintaining account states in Ethereum, as illustrated in Figure 6, requires transaction serialization. This condition limits high transaction throughput but is offset by transactions that require less storage capacity.

In the transactional models described before, we find significant differences in terms of transaction execution, which directly impacts storage requirements. For instance, the Bitcoin model can split one output to create new ones and merge multiple outputs into a smaller set, as shown in the Alice and Bob example. This flexibility means that the storage size of each transaction can vary depending on the number of outputs it manipulates. On the other hand, the account model manages the state in a serialized manner that is less storage-intensive but at less throughput. Each transaction updates the state of accounts directly, leading to a more predictable and often smaller storage footprint per transaction compared to the Bitcoin model.

To validate this analysis, we conduct an analytical study, comparing the two models by representing them as graphs (The details of the formal comparison of the two transactional models are available in Appendix A), and evaluate a particular case in the following subsection.

4.3. Transaction Sizes in Bitcoin and Ethereum

In this section, we analyze the Bitcoin and Ethereum blockchains. Our hypothesis based on the previous section is that the UTXO model requires more storage than the account model. For our comparison, we used a random sample of 10% of the transactions processed on each blockchain until 4 July 2023. This resulted in the analysis of 84,474,947 transactions in Bitcoin and 348,506,740 in Ethereum.

For data extraction, a set of specific tools and libraries were used: BlockSci version 0.7 [34], Geth version 1.12.0 [35], Python 3, along with the libraries Pandas, NumPy, Multiprocessing, and Matplotlib. The repository for reproducing the experiments can be found at: https://github.com/jdom1824/Unlocking-UTXO-transactional-patterns (accessed on 3 June 2024). The results obtained are visualized in the form of histograms, shown in Figures 7 and 8, to facilitate comparison. The *X*-axis represents the size of the transactions, while the *Y*-axis represents the number of transactions.

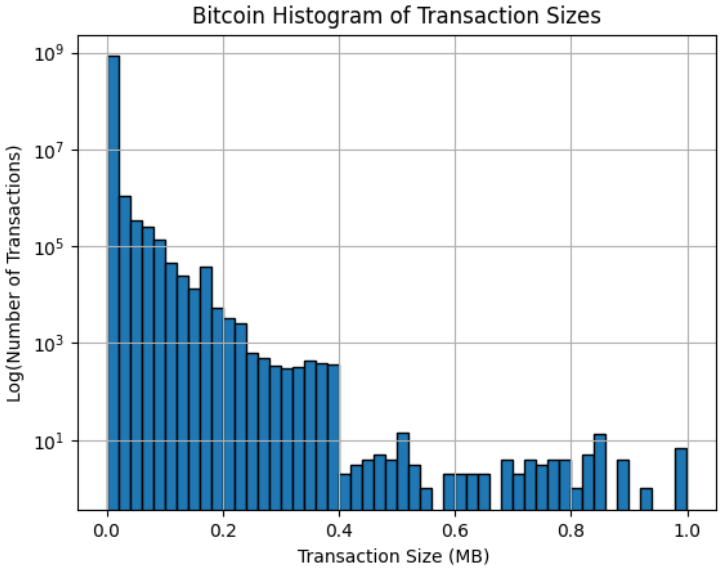

Figure 7. Histogram showing the distribution of Bitcoin transaction sizes on a logarithmic scale, compiled from a dataset of 84,474,947 transactions, highlighting the frequency of transaction sizes in megabytes.

When comparing the histograms, it is clear that the distribution of transactions in Bitcoin extends up to 1 MB. This is a significant size that reflects the robust nature of the UTXO model, as it can handle large transactions while resisting failures. In contrast, Ethereum operates differently. Only a small number of transactions in Ethereum reach a size of up to 0.3 MB. This is less than a third of the maximum observed in Bitcoin, indicating a more compact transaction size in Ethereum's model.

A closer look at the data reveals that most transactions in Ethereum are situated in the range of 0.13 MB. This is a narrower range compared to Bitcoin, where a wider distribution is observed, reaching up to 0.2 MB. This difference in distribution patterns between the two cryptocurrencies provides valuable insights into their respective transactional models.

As a result of these observations, the histograms suggest that the transactional model of Bitcoin implies a higher storage cost. This cost is not static; it is anticipated to escalate in line with the fragmentation of unspent outputs, as depicted in example (a) of Figure 5. This trend suggests that as Bitcoin usage increases, transactions increase storage requirements.

On the other hand, Ethereum presents a different scenario. It has a lower storage cost that is expected to remain constant within the same storage ranges. This stability is related to transaction serialization, indicating a more stable model for Ethereum in terms of storage. This has significant implications for the development of DApps on Ethereum.

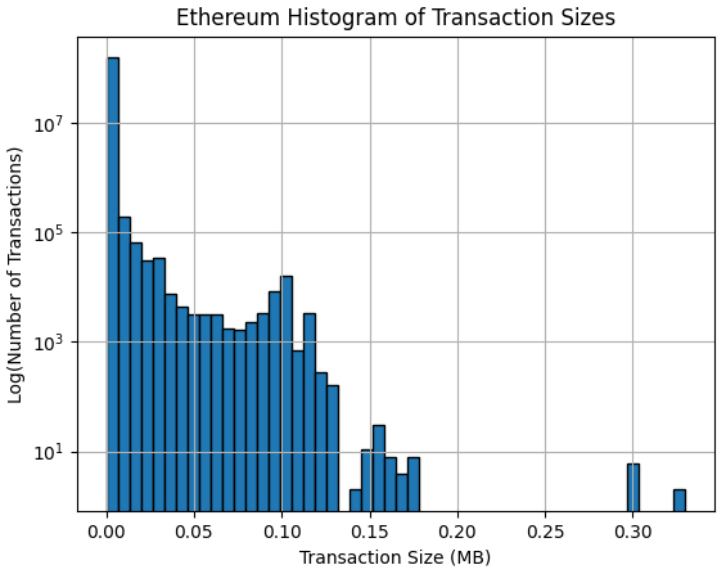

Figure 8. Histogram illustrating the size distribution of Ethereum transactions on a logarithmic scale, showing the variation in transaction sizes up to 0.3 megabytes.

4.4. Summary

This section analyzes the execution of the transactional models for both Bitcoin and Ethereum. We identified that the transactional model of Bitcoin is more flexible when selecting the available outputs to consume, while the Ethereum model presents simpler transactions that are easily programmable. The flexibility of the UTXO model makes it efficient when transferring value to users, while the account model is limited by serialization to update the state of the EVM.

We established the hypothesis that the UTXO transactional model incurs higher storage costs due to the splitting and consolidation of unspent outputs. We confirmed our hypothesis with an analytical study in Appendix A as well as the histograms shown in Figures 7 and 8. Although the UTXO model is storage-intensive, it also allows for significant transaction throughput. This is achieved by allowing multiple operations within a single transaction, providing flexibility between transaction throughput and storage, and showing the signs of the trade-off in the parameters.

In the following section, we focus on the UTXO transactional model, specifically on the model's flexibility to perform multiple operations. We delve deeper into transactional patterns to define the trade-off between storage parameters and transactional throughput.

5. Unlocking Transactional Patterns Based on Spent-By Relation

This section unlocks transactional patterns of the UTXO model to reveal the trade-off between transactional throughput and storage. To do this, we used abstractions from previous analyses and defined the spent-by relation. We then modified the cardinality of the spent-by relation using less than, greater than, and equal functions to observe three transactional patterns within the UTXO model: splitting, merging, and transferring. For clarity in this analysis, we proceed based on the premise that the number of nodes (η) within a permissionless blockchain system grows linearly.

5.1. Defining the UTXO Model as a DAG

The UTXO model is defined as a DAG. Formally, it is represented as a tuple $G = (V, R)$, where V is a finite set of vertices, and R is a set of edges, such that we have the following:

- The set of vertices represents the outputs of the UTXO model and is divided into two subsets $V = \Xi \cup \Theta$. Here, Ξ is the set of spent outputs, and Θ is the set of unspent outputs.
- The set of edges R is determined by the spent-by relation, which specifies how the Θ and Ξ are related.

5.2. Spent-By Relation "←"

To define the spent-by relation, we begin by partitioning the graph (G) into a subgraph, $H = (V', R')$, as illustrated in Figure 9, where $\Xi_s \subset \Xi$, $\Theta_s \subset \Theta$, such as $V' = \Xi_s \cup \Theta_s$.

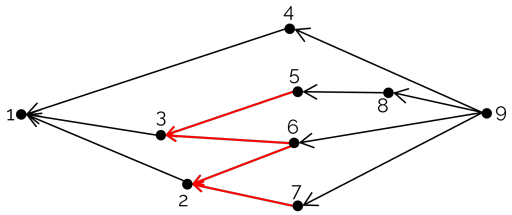

Figure 9. Visualization of a UTXO model's subset represented as a DAG, where the highlighted subgraph H delineates the relation between spent and unspent outputs within the system.

Let us define the set of edges, R', which satisfies the following properties:

- $R' = \{(x, y)$. This represents all pairs (x, y), where x and y are elements of the sets Ξ_s and Θ_s, respectively.
- $|R'| = |V'| - 1$. This means that the number of edges in R' is one less than the number of vertices in V'.

The spent-by relation defines the set of relations that exist between subsets of unspent outputs and spent outputs. Formally, we define the spent-by relation as a subset R' of the Cartesian product $\Xi_s \times \Theta_s$:

$$x \leftarrow y, \text{ where } x \in \Xi_s \text{ and } y \in \Theta_s \tag{1}$$

Based on the cardinality relation between Ξ_s and Θ_s, different transactional behaviors are observed: splitting, merging, and transferring. The splitting pattern occurs when a set of spent outputs is divided into a larger set of unspent outputs (i.e., $|\Theta_s| > |\Xi_s|$). The merging pattern manifests when multiple spent outputs are combined into a smaller number of unspent outputs (i.e., $|\Theta_s| < |\Xi_s|$). Lastly, the transferring pattern arises when each element in Ξ_s is linked precisely to one element in Θ_s ($|\Theta_s| = |\Xi_s|$), representing a one-to-one relation between spent and unspent outputs.

5.3. Unlocking Transactional Patterns

This section focuses on transactional patterns and introduces the relationship between throughput and storage parameters.

5.3.1. Splitting Pattern

To illustrate the splitting pattern, let us revisit the example of Alice and Bob, specifically referencing the scenario presented in Figure 5a. This pattern involves dividing one or several unspent outputs into smaller parts, as illustrated in Figure 10. However, it is important to highlight that we have generalized the splitting pattern by extending it to all scenarios where the set of unspent input values is greater than the set of spent output values.

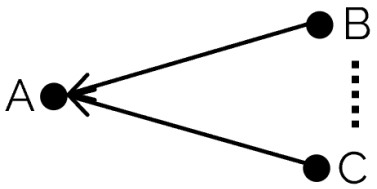

Figure 10. Splitting pattern, where a single input from A is divided into multiple outputs B, C, . . . , representing an n-number of possible outputs.

In the behavior of the splitting pattern, it is observed that the number of operations depends on a factor defined within the application. For instance, a single Bitcoin in an unspent output can be divided into up to 10^8 new outputs [36]. Therefore, to calculate the number of outputs per splitting pattern and its associated storage, we present the following definitions:

Definition 2. *(**Outputs per splitting pattern**) The number of outputs produced by a splitting pattern within a given time interval is quantified using two parameters: the splitting factor (κ_s) and the time interval (t), where $\kappa_s = |\Theta_s|$ and $|\Theta_s| > |\Xi_s|$. Consequently, the output rate per time interval can be expressed as follows:*

$$\sigma_s = \frac{\kappa_s}{t} \tag{2}$$

Definition 3. *(**Storage per output splitting pattern**) The storage generated by the splitting pattern is related to the average output size (τ), the number of outputs generated per time interval (σ_s), and the number of nodes in the system (η). This is represented as follows:*

$$\omega_s = \tau \sigma_s \eta \tag{3}$$

Note that the value of (κ_s) in Definition 2 is determined by each application, setting constraints on the number of new outputs. We operate under the assumption that κ_s is a very large number, and therefore, σ_s presents a high degree of transactional throughput. However, as indicated in Definition 3, there is a strong relation between transactional throughput and storage. This relation is only observable at the level of transaction models. Our observations reveal that as the number of outputs processed in a transaction increases, so does the storage cost on the nodes. Consequently, storage grows in proportion to transactional throughput.

To evaluate the maximum growth of storage, we employ a Big *O* notation. This indicates that the increase in storage, following the splitting pattern, is given by $O(\kappa_s \eta)$.

5.3.2. Merging Pattern

The merging pattern emerges from the consolidation of multiple outputs into a reduced set of unspent outputs, as illustrated in the example of Alice and Bob presented in the previous section, specifically in Figure 5b. The primary characteristic of the merging pattern lies in the reduction of the number of new outputs to a smaller set compared to the input values, establishing a balance with the splitting pattern. The abstraction of this pattern is illustrated in Figure 11. To calculate the number of outputs per merging pattern and the amount of storage used per output, we present the following definitions.

Definition 4. *(**Outputs per merging pattern**) In this definition, we use κ_m to represent the number of outputs generated by the merging pattern, where $\kappa_m = |\Theta_s|$ and $|\Theta_s| < |\Xi_s|$. Therefore, the number of outputs generated by the merging pattern equals the set of unspent outputs, which by definition are fewer than the number of spent outputs.*

$$\sigma_m = \frac{\kappa_m}{t} \tag{4}$$

Definition 5. *(**Storage per output merging pattern**) The average output size τ, the number of outputs generated per time interval (σ_m), and the number of nodes in the system (η) measure the storage generated by the merging pattern per time interval as follows:*

$$\omega_m = \tau \sigma_m \eta \tag{5}$$

Definitions 4 and 5 illustrate how the merging pattern improves the efficiency of future transactions. This improvement results from consolidating multiple outputs into a reduced set of unspent outputs, which reduces the processing constraint for subsequent transactions. As a result, less time and fewer computational resources are required to process and validate transactions, boosting the overall system efficiency. However, it is important to consider that defining the storage per output merging pattern suggests a similarity to the splitting pattern. We recognize that the average output size, τ, can vary significantly depending on the pattern or transaction type. We explore this variation further in Section 6. Storage growth, following the merging pattern, occurs at a rate of $O(\kappa_m \eta)$.

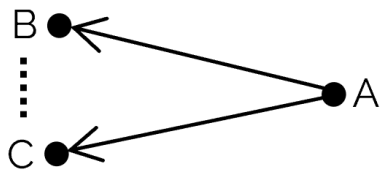

Figure 11. Merging pattern, where multiple outputs from nodes B, C, ..., converge into a single output at node A.

5.3.3. Transferring Pattern

The transferring pattern represents the exchange of ownership between parties without the need to engage in computational processing to split or merge unspent outputs. This pattern can be visualized in a scenario where an unspent output changes ownership through its inclusion as an input in a new transaction, generating a new output, as illustrated in Figure 5c.

The transferring pattern is a fundamental component in both the Bitcoin UTXO model and the Ethereum account model. In the UTXO model, it is characterized by the serialized tracing of unspent outputs, while in the account model, it updates the state of individual accounts or Ethereum addresses. Both models share the transferring pattern for managing transactions, as depicted in the abstraction shown in Figure 12.

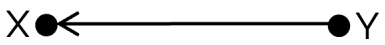

Figure 12. Transferring pattern, showing a direct relation from X to receiver Y.

An interesting feature of the transferring pattern is that only a one-to-one operation is carried out at each time interval. This structure has notable implications for both parameter storage requirements and transactional throughput.

Definition 6. *(**Storage per output transferring pattern**) The storage generated by the transferring pattern is related to the average output size (τ_a), the number of outputs generated per time interval (σ_t), and the number of nodes in the system (η). This is represented as follows:*

$$\omega_t = \tau \sigma_t \eta \tag{6}$$

Since a transaction in the transferring pattern is constrained by the non-concurrency of the operations, storage grows constantly. In terms of computational complexity, this means that the storage requirements for this pattern increase linearly with the number of nodes $O(\eta)$ in the network. This realization comes from the recognition that the transferring pattern is sufficient to represent the serialization process within the account model or UTXO model.

5.4. Relationship between Throughput and Storage

Transactional throughput refers to the system's capacity to process transactions over a time interval, and each transaction in environments such as Bitcoin can generate multiple outputs.

We consider the following parameters before defining the transactional throughput and its relationship with storage:

- **Outputs across transactional patterns** This parameter, denoted as κ, represents the total number of outputs generated by all transactional patterns (splitting, merging, and transferring). It is the sum of the outputs from each pattern, expressed as follows:

$$\kappa = \kappa_m + \kappa_t + \kappa_s \tag{7}$$

- **Number of outputs of all transactional patterns in a time interval:** This parameter, denoted as σ, represents the total number of outputs generated by all transactional patterns per time interval. It is calculated by dividing the total number of outputs κ by the time interval t, expressed as follows:

$$\sigma = \frac{\kappa}{t} \tag{8}$$

- **Average number of outputs per transaction:** This parameter, denoted as λ, represents the average number of outputs generated per transaction. It is calculated by dividing the total number of outputs κ by the total number of transactions Tx, expressed as follows:

$$\lambda = \frac{\kappa}{Tx} \tag{9}$$

Definition 7. *(**Transactional Throughput**) We define transactional throughput (tps) as the number of transactions processed per second. If σ is the total number of outputs generated in a time interval t, and λ is the average number of outputs per transaction, then transactional throughput is calculated as follows:*

$$tps \approx \frac{\sigma}{\lambda} \tag{10}$$

Definition 8. *(**Throughput-Storage Relationship**) The storage generated by each transactional pattern is related to the average output size (τ), the number of outputs generated per time interval (σ), and the number of nodes in the system (η). Therefore, the relation between the transactional throughput and storage is given by the following:*

$$\omega \approx \tau\sigma\eta \tag{11}$$

By increasing the transactional throughput (*tps*), we also increase the number of outputs per interval of time (σ) and, therefore, the required storage increases.

5.5. Summary

In this section, we unlock the transactional patterns inherent in the UTXO model. We formalize the UTXO model by representing it as a DAG and define the spent-by relation. We reveal the trade-off between transactional throughput and storage based on the definitions of each pattern, highlighting that storage growth is related to the number of new outputs generated. We analyze each pattern's contribution to storage size, employing Big *O* notation. The underlying premise is that the number of nodes in the permissionless

blockchain network increases at a linear rate. However, although analytically, the splitting and merging transactional patterns consume more storage, these results are not directly comparable due to our assumption of output size as a constant τ. In the following section, we delve deeper into this variable and define which pattern is most costly in storage and which provides more flexibility in the throughput.

6. Experimental Comparison of Storage Costs in UTXO Transactional Patterns

This section analyzes the storage cost of each pattern to identify which is higher and which provides greater flexibility in the storage/throughput trade-off.

In the theoretical analysis that we previously conducted, we used a constant τ_a for the transferring, merging, and splitting transactional patterns. For this experimental study, we used the entire Bitcoin blockchain as our dataset, examining a total of 791,800 blocks to determine the storage of each pattern. Figure 13 shows the experimental framework for our analysis using Bitcoin Core version 0.22 [37]. We synchronized a complete Bitcoin node up to 4 July 2023 and extracted data for further analysis using BlockSci version 0.7.0. After extracting the data, we filtered the dataset based on transaction patterns and converted it into graphical representations to enhance the clarity and interpretability of the results discussed in this section. Derived from this work, we have created a database containing 800 million transactions, which can be used to replicate the experiments in [38].

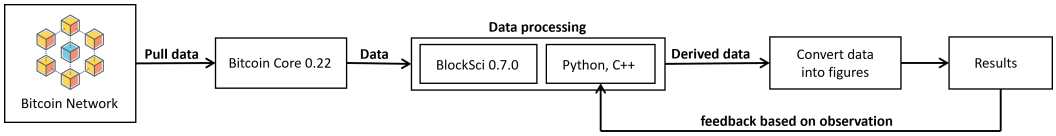

Figure 13. Flowchart of the experimental framework used for analyzing transactional patterns in the UTXO Model, starting from data extraction using Bitcoin Core 0.22, processing with BlockSci 0.7.0 and Python 3/C++, to the final stage of converting data into figures for result interpretation and feedback iteration.

As mentioned before, the initial step taken with the dataset involved filtering and classifying Bitcoin transactions. This classification results in the distribution of transactional patterns within Bitcoin and is represented in a pie chart, as shown in Figure 14. We observed that the splitting pattern is the most frequent in Bitcoin, accounting for 64.6% with a total of 545,585,796 transactions. This trend emerges because Bitcoins are generated through the Coinbase transaction, which includes a UTXO with a significant amount of Bitcoin. Due to the high dollar value of each Bitcoin, their utilization likely begins with a division.

The transferring pattern accounts for 22.1% of classified transactions, totaling 186,881,657. We can assert that this is the second most utilized pattern in Bitcoin. The reason is that in Bitcoin, a fee is levied based on the storage consumed by the transaction. Since this pattern is the least storage-intensive, it is the second most common.

The merging pattern accounts for 13.3% of transactions, amounting to 112,107,603. From these data, we infer that the consolidation of unspent outputs is a more storage-intensive process. We assume that the available output for expenditure must encompass the causal history of previous transactions.

The classification depicted in Figure 14 reflects the most used patterns in Bitcoin. From this, we discern an indication suggesting that the merging pattern is the most storage-intensive. We then analyze each transaction pattern individually, considering the number of outputs against storage size. This clarifies the storage difference between the splitting and merging patterns. Moreover, we confirm that the least storage-intensive transaction pattern is transferring.

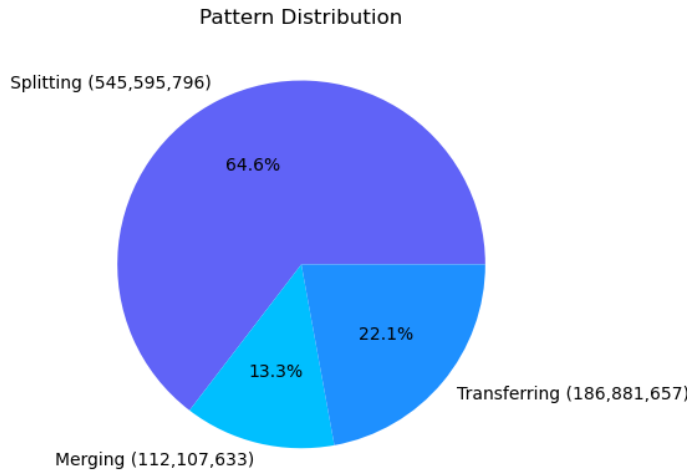

Figure 14. Pie chart showing the relative distribution of splitting, merging, and transferring patterns within Bitcoin, with numerical and percentage breakdowns for each category.

6.1. Storage Cost in Splitting Pattern

Figure 15 provides a graphical illustration of transactions classified under the splitting pattern. The *X*-axis represents the size of the transactions in bytes, whereas the *Y*-axis represents the number of outputs used in each transaction. Through an in-depth analysis of the data density and distribution depicted in the chart, we confirm our initial observation that the splitting pattern is dominant within Bitcoin.

Concerning the relation between the number of outputs and storage costs, we identified transactions labeled as splitting, which recorded up to 15,000 outputs in a single transaction. In terms of storage, this transaction has demanded up to 0.5 MB. Nevertheless, the transactions tend to fall within a range of up to 4000 outputs with a storage requirement that is close to 0.2 MB.

We highlight the significance of the splitting pattern in Bitcoin. While it is the most common transactional pattern, and some transactions demand substantial storage resources, the overall trend remains moderate. It is important to note that one Bitcoin is split into up to 100 million parts, making this thorough analysis of the pattern crucial to guide future research efforts within the Bitcoin network.

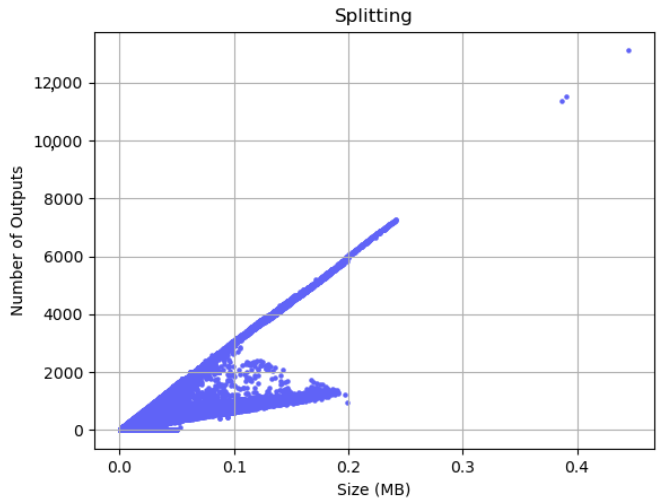

Figure 15. Scatter plot correlating transaction size in megabytes (MB) to the number of outputs for transactions that follow the splitting pattern, where each point represents a single transaction.

6.2. Storage Cost in Transferring Pattern

Figure 16 provides a graphical illustration of transactions classified under the transferring pattern. Based on the spent-by relation, this pattern contains transactions that maintain a one-to-one operation within the set of outputs. In the graph, the *X*-axis represents the size of the transactions in bytes, while the *Y*-axis indicates the number of outputs used.

It is observed that some transactions reach up to 2000 outputs, with a storage cost of 0.15 MB. However, the overall trend revolves around transactions using approximately 500 outputs, with a storage requirement of about 0.05 MB.

In addition, in Figure 16, two distinct point distributions are revealed, each representing a specific transaction type. Upon analysis, it is meaningful that certain transactions with a larger number of outputs have a lower storage cost, especially in the 0.05 to 0.06 MB range. This variability in storage arises from the diversity of transaction types in Bitcoin, which includes standard transactions, Multisig transactions [39], Pay-to-Script-Hash (P2SH) transactions [40], SegWit transactions [41], CoinJoin transactions [42], and time-locked transactions [43]. Each type has its unique storage characteristics and requirements, reflecting the variety of transactions observed in the graph.

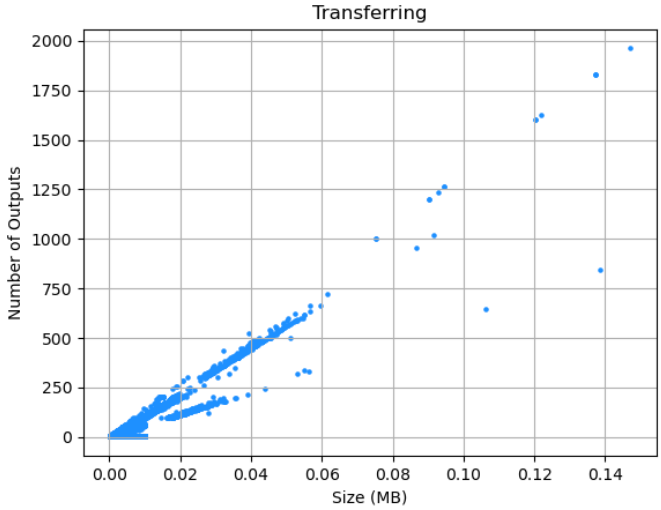

Figure 16. Scatter plot showing the relation between transaction size (MB) and the corresponding number of outputs for the transferring pattern, maintaining a one-to-one spent-by relation, where each data point represents a single transaction with an equal number of inputs and outputs.

6.3. Storage Costs in the Merging Pattern

Figure 17 provides a classification of transactions under the merging pattern. In this chart, the X-axis represents the volume of the transactions in megabytes (MB), while the Y-axis quantifies the number of outputs involved. It is noteworthy that several transactions reach up to 1 MB, which corresponds to the maximum capacity of a Bitcoin block before the SegWit implementation, with an output range oscillating between 6000 and 7500. However, transactions within this pattern fall within a range of approximately 2500 outputs, consuming storage close to 0.2 MB.

Analogous to previous figures, some transactions have a storage distribution that deviates from the classification of the merging pattern. Note that Bitcoin offers a variety of transaction types. This diversity is interesting for future studies and possible classification of patterns in different Bitcoin transaction types [44].

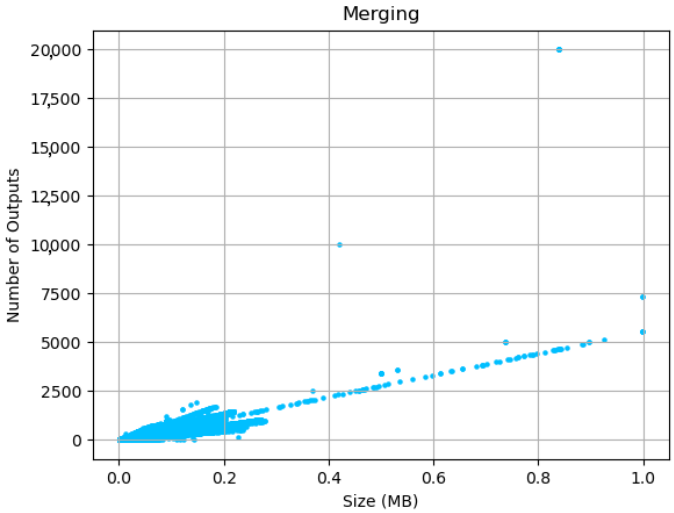

Figure 17. Scatter plot showing the relation between transaction size in megabytes (MB) and the number of outputs for transactions characterized by the merging pattern, illustrating the consolidation of multiple inputs into fewer outputs.

6.4. Analysis of Transaction Pattern in Storage/Throughput Flexibility

In our detailed review of the three patterns, we observed that the transferring pattern has the lowest storage requirement, rating it as the second most common pattern in Bitcoin. The splitting pattern is the most common and offers the best trade-off between transactional throughput and storage. The merging pattern supports operations that consolidate outputs on the order of thousands but require more storage. However, the storage cost for each pattern varies depending on the structure. For example, a structure with a higher number of spent outputs than unspent outputs is more costly in terms of storage because it is necessary to prove ownership of the coins by unlocking the transaction script, which requires a digital signature, as shown in Table 2. Further comparison between the structures of spent and unspent outputs, depicted in Tables 2 and 3, illustrates the different storage requirements.

Table 2. Spent output in a regular Bitcoin transaction.

Attribute	Description	Size
PrevTxid	Hashed ID of the to-be-used transaction	32 bytes
Output Index	Number of the output in the transaction	4 bytes
SigScript	Signature data to unlock spent output	Variable
Sequence	Transaction sequence number	4 bytes

Table 3. Unspent output in a regular Bitcoin transaction.

Attribute	Description	Size
PkScript	Script that sets the conditions to unlock funds	Variable
Value	Satoshi Amount	8 bytes

6.5. Summary

In this section, we found that in the UTXO model, there is no fixed storage value for spent and unspent outputs; this varies depending on the transactional pattern and types of transactions. We observed that the splitting pattern offers the best trade-off between throughput and storage, allowing millions of operations in a single transaction while keeping the storage low. However, this benefit is offset by the merging pattern, which consolidates these operations into transactions that, although more storage-intensive, reduce the number of outputs and prevent overflow in processing. Finally, we conclude that the key to achieving storage scalability in a permissionless blockchain system resides

in proposing strategies that optimally trade off the relationship between throughput and storage at the transaction pattern level.

7. Discussion

This research was the first to highlight the importance of the relationship between throughput and storage efficiency, setting the stage for future research on achieving high transactional throughput without sacrificing storage efficiency.

In the current state of the art, different approaches tend to focus on throughput at the expense of storage, or vice versa. For example, while techniques such as sharding and off-chain improve throughput, they also introduce storage challenges. Similarly, storage reduction methods reduce transactional throughput. Our approach shows that it is possible to achieve a balance between the two parameters. For example, Section 6.1 reveals that the splitting pattern in the UTXO model maintains a high number of operations while using low storage consumption. Thus, exploring techniques based on generating this pattern more intensely instead of others will be favorable in terms of storage requirements. This insight paves the way for new blockchain designs that hold this trade-off, leading to a more scalable blockchain.

7.1. Practical Implications of Transactional Patterns

Unlocking transactional patterns to abstract transactions in a granular manner showcases its applicability across several blockchain research. For instance, the direct relation between inputs and outputs that our model describes enhances traceability analyses. In high-frequency trading environments where private blockchains are used, and storage constantly grows, the splitting patterns could increase throughput by allowing transactions to be executed in parallel. Lastly, new types of transactions could be proposed based on the identified transactional patterns. These innovations could enhance privacy and security in blockchain environments.

7.2. Discussion of Experimental Results

The experimental comparison based on the classification of transactions of 791,800 blocks shows how each pattern grows in storage requirements according to the number of outputs. For example, the splitting pattern, which represents 64.6% of the transactions, shows that its average storage growth per number of outputs is 32 bytes. This flexibility to increase the number of operations at a relatively low storage cost makes this pattern storage efficient.

The transferring pattern, which comprises 22.1% of the transactions, requires around 0.05 MB for approximately 500 outputs, or about 100 bytes per output. This sets it in the intermediate in terms of storage efficiency.

On the other hand, the merging pattern, which represents 13.3% of the transactions, involves the consolidation of multiple inputs into fewer outputs, which is inherently more storage-intensive. This consolidation pattern has an average output size of 128 bytes. Although it is crucial for managing and reducing the number of UTXOs in the system, it also introduces higher storage costs, with transactions that can reach up to 1 MB.

7.3. Future Research

Future research explores models that delineate the relationships among transactional throughput, storage, latency, availability, and reachability. Additionally, future studies investigate different transaction types in Bitcoin to develop methods to optimize storage efficiency.

One strategy for future work is to maintain the balance between the set of outputs in a transaction by identifying transactional patterns. For example, a set of transactions in the mempool could be grouped according to the splitting and merging pattern into a single transaction, similar to a CoinJoin transaction, thus reducing storage requirements and allowing more transactions to be processed per block, increasing throughput.

We anticipate that any method that seeks to increase transactional throughput will also need to consider the storage requirements. Future suggestions from this study could explore the fragmentation of the blockchain through transactional patterns to manage space and carefully increase throughput. We invite other researchers to use the databases [38] and tools shared in this study to analyze blockchains based on the UTXO model, such as Litecoin, Dogecoin, and Cardano. Future work with these tools will aim to identify transactional patterns of these blockchains and compare them with this study to improve the storage scalability of the system.

8. Conclusions

This research focuses on a permissionless public blockchain and reveals the trade-off between the storage and transactional throughput parameters. We unlocked the transactional patterns of the Bitcoin and Ethereum transactional models and found a direct relation between transactional throughput and storage. We defined the spent-by relation that reveals transactional patterns within the UTXO model, facilitating the categorization of Bitcoin and Ethereum transactions. After performing a detailed analysis of the storage growth corresponding to each pattern, we found that the UTXO model requires more storage overhead compared to the account model. This was done by abstracting the transactional patterns and evaluating each pattern in terms of storage growth using Big *O* notation, assuming that the set of nodes that belongs to the permissionless blockchain network grows linearly. We have successfully encapsulated the transactional behavior in both Bitcoin and Ethereum networks. Our results highlight the need to consider the relationship between throughput and storage to achieve scalability in blockchain storage.

Author Contributions: Conceptualization, D.M., L.M.X.R.-H., J.C.P.-S. and S.E.P.-H.; methodology, D.M., L.M.X.R.-H., J.C.P.-S. and S.E.P.-H.; software, D.M.; validation, D.M., L.M.X.R.-H., J.C.P.-S. and S.E.P.-H.; formal analysis, D.M., L.M.X.R.-H., J.C.P.-S. and S.E.P.-H.; investigation, D.M., L.M.X.R.-H., J.C.P.-S. and S.E.P.-H.; data curation, D.M.; writing—original draft preparation, D.M.; writing—review and editing, D.M., L.M.X.R.-H., J.C.P.-S. and S.E.P.-H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Council for Humanities, Sciences, and Technology (CONAHCYT); grant number 788159.

Data Availability Statement: All data is available in the following link: https://doi.org/10.7910/DVN/6V8HRL.

Acknowledgments: We would like to express our deepest gratitude to the National Supercomputing Laboratory of the INAOE for the support, which made this research possible. Their provision of high-performance computing resources was instrumental in the successful outcome of this research article.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DApps	decentralized applications
DAG	directed acyclic graph
P2P	peer-to-peer
UTXO	unspent transaction output
TPS	transactions per second
CUB	consensus unit-based
EVM	Ethereum virtual machine
SE-Chain	secure and efficient chain
SASLedger	secure, accelerated scalable ledger
Jidar	jigsaw-like data reduction
SegWit	segregated witness

Appendix A. Formal Comparison of the Transactional Models of Bitcoin and Ethereum

In Ethereum, as opposed to Bitcoin, transactions are processed in a specific sequential order. This serialization is critical as it significantly influences the execution of smart contracts and the global state of the system. In this appendix, we formalize the functions related to transactional ordering in Bitcoin and Ethereum to understand the implications of processing transactions with serialized or concurrent operations on storage and transactional throughput.

The formalism focuses on two aspects: the execution of the operation in both transactional models and the validation of transactions in a blockchain environment. We aim to compare Ethereum's account-based model with Bitcoin's UTXO model, both of which are the subjects of this paper.

Appendix A.1. Formalization of Transaction Execution in Ethereum

To understand the serialized execution of Ethereum's transactions, we formalize the execution of simple transactions, the execution of smart contracts, the global state of the system, and the blockchain architecture. Additionally, we introduce the formalization of a state validation function that Ethereum uses to ensure a consistent state across the nodes participating in Ethereum.

Appendix A.1.1. Defining the Fundamental Sets

(a) ***T:*** This set represents all possible transactions in the system. A transaction is a simple token transfer or the invocation of a smart contract.
(b) ***B:*** This set represents the blocks in the blockchain. In Ethereum, each block contains an ordered set of transactions that have been validated and confirmed by the network.
(c) ***S:*** This is the global state of the system, which includes information such as the balance of each account, the code of smart contracts, and other relevant global data.
(d) ***C:*** This set represents all smart contracts deployed in the system. A smart contract is an autonomous program that runs on the blockchain.

Appendix A.1.2. Defining Functions

The functions we formalize below focus on how the system's state transitions with the execution of transactions. This approach is crucial for understanding the decentralized Ethereum virtual machine (EVM) of Ethereum at a granular level, including the impact that serialization has on state transitions.

(a) ***State transition function:*** $\text{apply}(S, t) \rightarrow S'$ This function takes an initial state (S) and a transaction (t) and returns a new state (S'). If the transaction is invalid, S' is equal to S.
(b) ***Validation function:*** $\text{isValid}(S, t) \rightarrow \{\text{True, False}\}$ This function checks if a transaction (t) is valid given a state (S). This involves checking if the sender has enough funds to complete the transaction or whether the transaction complies with certain smart contract rules.
(c) ***Sequentiality function:*** $\text{blockSeq}(B) = [t_1, t_2, \ldots, t_n]$ This function extracts the serialized order of transactions in a block, B.
(d) ***State update function:*** $S_{new} = apply(\ldots apply(apply(S, t_1), t_2) \ldots, t_n)$ It shows how the global state (S) is updated by sequentially applying each transaction in B, taking into account validation.

Appendix A.2. Simple Transaction in Ethereum

Initial State S**:** Assume we have an initial state, where account A has 4-ether and account B has 3-ether.

$$S = \{(A, 4), (B, 3)\}$$

Transaction t**:** Account A wants to send 3-ether to account B and initiates the following transaction:

$$t = \text{transfer}(A, B, 3)$$

Validation isValid (S, t)**:** Prior to including the transaction in a block, i.e., before executing it, the system checks whether the transaction is valid. In this case, as A has sufficient funds, the transaction is valid.

$$\text{isValid}(S, t) = \text{True}$$

State Transition Function apply $(S, t) \rightarrow S'$**:** The global system state is updated using the function apply.

$$S' = \text{Apply}(S, t) = \{(A, 1), (B, 6)\}$$

Here, the state is updated to reflect the value transfer between the two accounts.

Appendix A.3. Smart Contract Execution in Ethereum: Success and Failure Scenarios

Smart contract C**:** Consider a smart contract that doubles incoming ether with a 3-ether minimum and an initial balance of 2-ether.

$$C(x) = \begin{cases} 2x & \text{if } x \geq 3 \\ \text{Fail} & \text{otherwise} \end{cases}$$

Appendix A.3.1. Success Scenario

Initial State S: Assume we have an initial state, where account A has 5-ether and the contract C has 2-ether.

$$S = \{(A, 5), (C, 2)\}$$

Transaction t: Account A wants to send 3-ether to C using the "double" function.

$$t = \text{double}(A, C, 3)$$

Validation isValid(S, t): A has sufficient funds to send 3-ether, the transaction is valid.

$$\text{isValid}(S, t) = \text{True}$$

State Transition Function apply$(S, t) \rightarrow S'$: The state updates to reflect the "double" operation.

$$S' = \text{Apply}(S, t) = \{(A, 2), (C, 2 + 2 \times 3)\}$$

Appendix A.3.2. Failure Scenario

Transaction t': Account A wants to send 2-ether to C using the "double" function.

$$t' = \text{double}(A, C, 2)$$

Validation isValid (S, t'): C requires a minimum of 3-ether for the "double" function, the transaction is invalid.

$$\text{isValid}(S', t') = \text{False}$$

State Transition Function apply $(S', t') \rightarrow S'$: Because the transaction is invalid, the state remains unchanged.

$$S'' = \text{Apply}(S', t') = S'$$

Appendix A.4. Sequentiality Function in Ethereum Transactions

As observed in previous examples, Ethereum maintains a global view of the state of the EVM to execute transactions. However, to ensure the consistency of state, transactions are executed in a serialized manner within a single block. Following this, we illustrate this process with an example featuring a block filled with multiple transactions.

$$\text{blockSeq}(B) = [t_1, t_2, t_3]$$

where $t_1 = \text{transfer}(A, B, 1)$, $t_2 = \text{transfer}(A, C, 1)$ and $t_3 = \text{transfer}(A, D, 1)$, the state transition is as follows:

$$S' = \text{apply}(S, t_1)$$
$$S'' = \text{apply}(S', t_2)$$
$$S''' = \text{apply}(S'', t_3)$$

Given an initial state $S = \{(A, 4), (B, 0), (C, 0), (D, 0)\}$, after executing transaction t_1, the new state S' becomes $\{(A, 3), (B, 1), (C, 0), (D, 0)\}$. Following transaction t_2, the state updates to $S'' = \{(A, 2), (B, 1), (C, 1), (D, 0)\}$. Finally, executing t_3 results in a state $\{(A, 1), (B, 1), (C, 1), (D, 1)\}$.

Based on the previously illustrated transactional examples and the sequential representation of states, we assert that serialization is a strong yet necessary constraint to maintain a distributed virtual machine. We model these state changes using the concept of a directed acyclic simple path. This term refers to a directed acyclic graph (DAG) containing precisely a set of vertices linked by edges, where there is a single path between the vertices, as shown in Figure A1.

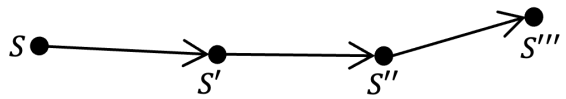

Figure A1. The illustration shows a directed acyclic graph where each vertex S, S', S'', S''' symbolizes a state in a distributed virtual machine environment.

Appendix A.5. Formalization of Transaction Execution in Bitcoin

As we observed before, in Ethereum each transaction modifies a global state that is a summary of all accounts, smart contracts, and other digital assets within the network. In contrast, in Bitcoin's UTXO model, the idea of an explicit global state is absent. Instead of maintaining accounts with updatable states, Bitcoin operates on a dynamic set of unspent transaction outputs. Each unspent output within the system is conceptualized as a microstate. When a transaction is executed, it consumes an unspent output to create a new one, adding the new output to a global set of UTXOs available for future transactions.

Appendix A.5.1. Definition of Fundamental Sets

(a) ***T:*** Encapsulates all transactions within the system. *A* transaction is defined as a simple token transfer operation.

(b) ***B:*** This set comprises all blocks in the blockchain. In the context of Bitcoin, each block contains a set of transactions that are validated and confirmed by the network nodes.

(c) ***USet:*** Refers to a set of all available unspent transaction outputs (UTXOs) at any given time. USet is an abstraction of the set of microstates in Bitcoin.

Appendix A.6. Simple Transaction in Bitcoin

Initial $USet$: Assume we have an initial set of unspent outputs, wherein each output belongs to a distinct owner as shown in Figure A2 with the different colors of the vertices.

$$USet = \{(A,5),(B,2),(C,7)\}$$

Transaction t: The owner of unspent output A wants to transfer BTC 1 to the other two owners.

$$t = \text{transfer}\{(A,D,1),(A,E,1),(A,F,3)\}$$

Note that when the owner of A elects to spend this UTXO, new outputs are created even for the remaining balance. For instance, in the new unspent outputs D and E, BTC 1 is transferred to each, and in output F, the remaining Bitcoins are returned.

Validation $isValid$: Here, we verify that A has enough Bitcoins to carry out the transaction.

$$\text{isValid}(USet,t) = \text{True} \quad \text{if} \quad (A,x) \in USet \,\text{and}\, x \geq 2$$

Transition Function $apply\ (Uset,t) \rightarrow USet'$: If the transaction is valid, we update the set.

$$USet' = \text{apply}(USet,t) \rightarrow$$
$$\{(B,2),(C,7),(D,1),(E,1),(F,3)\}$$

In the previous example of a Bitcoin transaction, we observed a specific method for updating the set of unspent outputs. The first key observation is that the UTXO model operates efficiently without the need for a comprehensive view of the $USet$ to add and consume unspent outputs. Additionally, we observe that this model enables the execution of multiple operations within a single transaction. Notably, the outputs used in a transaction are consumed to create new ones. If there is any balance, the user must create a new output to return the balance.

Figure A2 illustrates an abstract of the transition of microstates within $USet$. It is observed that output $(A,5)$ is consumed to generate new outputs directed toward other owners. This highlights a unique aspect of Bitcoin: it does not maintain a global state but rather operates as a collection of microstates.

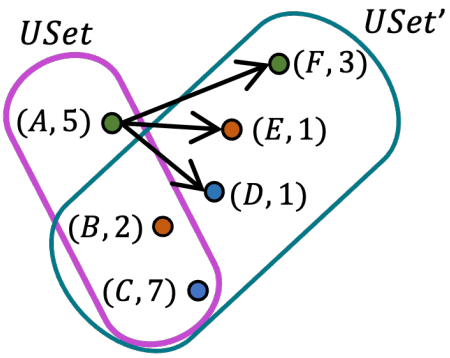

Figure A2. This figure demonstrates the transition of state from $USet$ to $USet'$ upon execution of transaction t. Each state, represented by a vertex (e.g., $(A,5)$), indicates an ownership state with an associated value.

Appendix A.7. Comparative Complexity Analysis in Transactional Models

To conduct the comparative analysis, we use the Big O notation to model the maximum number of operations generated by a transaction during a state change in each transactional model.

In the UTXO model, as we have previously seen, a single transaction t can theoretically divide a UTXO into n new UTXOs. If we have m transactions in a state change, then the maximum number of new operations is $O(m^n)$.

In contrast, the account model allows each transaction t to generate at most a single operation. In a state change with m transactions, the maximum number of operations is m, and the complexity is $O(m)$, considering that each transaction can perform one operation or modify a smart contract, as shown in previous examples.

It is crucial to note that the abstraction of these models is used to compare them in terms of operations per state transition and, therefore, does not capture the complexity of more advanced transactions in Bitcoin or Ethereum.

From the perspective of Big O notation, we assert that the account model has a lower computational complexity to process a number of operations in a state transition, while the UTXO model has a higher complexity. It is noted that these calculations are theoretical and do not consider practical limitations, such as block size or the maximum number of divisions of a UTXO in Bitcoin. However, this abstraction allows us to conclude that the UTXO model is more efficient for generating a large number of operations in a state transition, although this performance comes at a higher storage cost. On the other hand, Ethereum is less costly in terms of storage, but the number of operations per state transition is limited by its serialization.

References

1. Belotti, M.; Božić, N.; Pujolle, G.; Secci, S. A Vademecum on Blockchain Technologies: When, Which, and How. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 3796–3838. [CrossRef]
2. Neudecker, T.; Hartenstein, H. Network Layer Aspects of Permissionless Blockchains. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 838–857. [CrossRef]
3. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; Caro, A.D.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. *arXiv* **2018**, arXiv:1801.10228.
4. Johnston, D.; Yilmaz, S.O.; Kandah, J.; Bentenitis, N.; Hashemi, F.; Gross, R.; Wilkinson, S.; Mason, S. DApps. 2014. Available online: https://github.com/DavidJohnstonCEO/DecentralizedApplications (accessed on 3 June 2024).
5. Blockchair. Blockchain Size. 2023. Available online: https://blockchair.com/bitcoin (accessed on 3 June 2024).
6. Etherscan. Blockchain Size. 2023. Available online: https://etherscan.io/ (accessed on 3 June 2024).
7. Wikipedia. SegWit. 2024. Available online: https://es.wikipedia.org/wiki/SegWit (accessed on 3 June 2024).
8. Wackerow, P. BLOCKS. 2024. Available online: https://ethereum.org/en/developers/docs/blocks/ (accessed on 3 June 2024).
9. Poon, J.; Dryja, T. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. Technical Report, Lightning Labs. 2016. Available online: https://lightning.network/lightning-network-paper.pdf (accessed on 3 June 2024).
10. Kappos, G.; Yousaf, H.; Piotrowska, A.M.; Kanjalkar, S.; Delgado-Segura, S.; Miller, A.; Meiklejohn, S. An Empirical Analysis of Privacy in the Lightning Network. In Proceedings of the Financial Cryptography and Data Security—25th International Conference, FC 2021, Virtual Event, 1–5 March 2021; Revised Selected Papers, Part I; Borisov, N., Díaz, C., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germnay, 2021; Volume 12674, pp. 167–186. [CrossRef]
11. Karaarslan, E.; Konacakli, E. Data Storage in the Decentralized World: Blockchain and Derivatives. *arXiv* **2020**, arXiv:2012.10253.
12. Worley, C.; Skjellum, A. Blockchain Tradeoffs and Challenges for Current and Emerging Applications: Generalization, Fragmentation, Sidechains, and Scalability. In Proceedings of the 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Halifax, NS, Canada, 30 July–3 August 2018; pp. 1582–1587. [CrossRef]
13. Luu, L.; Narayanan, V.; Zheng, C.; Baweja, K.; Gilbert, S.; Saxena, P. A Secure Sharding Protocol For Open Blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS'16, Vienna Austria, 24–28 October 2016; pp. 17–30. [CrossRef]
14. Kokoris-Kogias, E.; Jovanovic, P.; Gasser, L.; Gailly, N.; Syta, E.; Ford, B. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 20–24 May 2018; pp. 583–598. [CrossRef]
15. Zamani, M.; Movahedi, M.; Raykova, M. RapidChain: Scaling Blockchain via Full Sharding. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS'18, Toronto, ON, Canada, 15–19 October 2018; pp. 931–948. [CrossRef]
16. Xu, Z.; Han, S.; Chen, L. CUB, a Consensus Unit-Based Storage Scheme for Blockchain System. In Proceedings of the 2018 IEEE 34th International Conference on Data Engineering (ICDE), Paris, France, 16–19 April 2018; pp. 173–184. [CrossRef]

17. Dai, X.; Xiao, J.; Yang, W.; Wang, C.; Jin, H. Jidar: A Jigsaw-like Data Reduction Approach without Trust Assumptions for Bitcoin System. In Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), Dallas, TX, USA, 7–10 July 2019; pp. 1317–1326. [CrossRef]
18. Sun, H.; Pi, B.; Sun, J.; Miyamae, T.; Morinaga, M. SASLedger: A Secured, Accelerated Scalable Storage Solution for Distributed Ledger Systems. *Future Internet* **2021**, *13*, 310. [CrossRef]
19. Jia, D.Y.; Xin, J.C.; Wang, Z.Q.; Lei, H.; Wang, G.R. SE-Chain: A Scalable Storage and Efficient Retrieval Model for Blockchain. *J. Comput. Sci. Technol.* **2021**, *36*, 693–706. [CrossRef]
20. Li, C.; Zhang, J.; Yang, X.; Youlong, L. Lightweight blockchain consensus mechanism and storage optimization for resource-constrained IoT devices. *Inf. Process. Manag.* **2021**, *58*, 102602. [CrossRef]
21. Zahnentferner, J. An Abstract Model of UTxO-Based Cryptocurrencies with Scripts. 2018. Available online: https://eprint.iacr.org/2018/469 (accessed on 3 June 2024).
22. Buterin, V. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. 2014. Available online: https://ethereum.org/en/whitepaper/ (accessed on 3 June 2024).
23. Jeyakumar, S.; Hou, Z.; Yugarajah, A.; Palaniswami, M.; Muthukkumarasamy, V. Visualizing Blockchain Transaction Behavioural Pattern: A Graph-based Approach. *TechRxiv* **2023**. [CrossRef]
24. Narula, N.; Dryja, T. Cryptocurrency Engineering and Design. MIT OpenCourseWare. 2018. Available online: https://ocw.mit.edu/courses/media-arts-and-sciences/mas-s62-cryptocurrency-engineering-and-design-spring-2018/ (accessed on 3 June 2024).
25. Antonopoulos, A.M.; Wood, G. Smart Contracts and Solidity. In *Mastering Ethereum: Building Smart Contracts and DApps*, 1st ed.; O'Reilly Media: Sebastopol, CA, USA, 2018; Chapter 6; pp. 100–120.
26. Zheng, G.; Gao, L.; Huang, L.; Guan, J. *Ethereum Smart Contract Development in Solidity*; Springer: Singapore, 2021; Chapter 6. [CrossRef]
27. Etherscan. web3.eth—Call. 2023. Available online: https://etherscan.io/ (accessed on 3 June 2024).
28. Chakravarty, M.M.T.; Chapman, J.; MacKenzie, K.; Melkonian, O.; Jones, M.P.; Wadler, P. The Extended UTXO Model. In Proceedings of the Financial Cryptography and Data Security—FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, 14 February 2020; Revised Selected Papers; Bernhard, M., Bracciali, A., Camp, L.J., Matsuo, S., Maurushat, A., Rønne, P.B., Sala, M., Eds.; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2020; Volume 12063, pp. 525–539. [CrossRef]
29. Dai, P.; Mahi, N.; Norta, A. Smart-Contract Value-Transfer Protocols on a Distributed Mobile Application Platform. 2017. Available online: https://api.semanticscholar.org/CorpusID:36981890 (accessed on 3 June 2024).
30. Lombrozo, E.; Lau, J.; Wuille, P. Segregated Witness (Consensus Layer). Bitcoin Improvement Proposal. 2015. Available online: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki (accessed on 3 June 2024).
31. Benet, J. IPFS—Content Addressed, Versioned, P2P File System. *arXiv* **2014**, arXiv:1407.3561.
32. Swarm Team. Swarm—Storage and Communication Infrastructure for a Self-Sovereign Digital Society. *White Paper*. 2023. Available online: https://www.ethswarm.org/swarm-whitepaper.pdf (accessed on 3 June 2024).
33. Balaji, S.B.; Krishnan, M.N.; Vajha, M.; Ramkumar, V.; Sasidharan, B.; Kumar, P.V. Erasure coding for distributed storage: An overview. *Sci. China Inf. Sci.* **2018**, *61*, 100301. [CrossRef]
34. Kalodner, H.; Möser, M.; Lee, K.; Goldfeder, S.; Plattner, M.; Chator, A.; Narayanan, A. BlockSci: Design and applications of a blockchain analysis platform. In Proceedings of the 29th USENIX Security Symposium (USENIX Security 20), Boston, MA, USA, 12–14 August 2020; USENIX Association: Berkeley, CA, USA, 2020; pp. 2721–2738.
35. Wilcke, J. go-ethereum. 2013. Available online: https://github.com/ethereum/go-ethereum (accessed on 3 June 2024).
36. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 3 June 2024).
37. contributors, W. Bitcoin Core—Wikipedia, The Free Encyclopedia. 2024. Available online: https://en.wikipedia.org/wiki/Bitcoin_Core (accessed on 3 June 2024).
38. Melo, D.; Rodríguez-Henríquez, L.M.X.; Hernández, S.P.; Pérez-Sansalvador, J.C. Replication Data for: Unlocking Blockchain UTXO Transactional Patterns. 2024. Harvard Dataverse V7. Available online: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/6V8HRL (accessed on 3 June 2024).
39. Bitcoin Forum—Index. 2024. Available online: https://bitcointalk.org/ (accessed on 3 June 2024).
40. Perez-Sola, C.; Delgado-Segura, S.; Herrera-Joancomarti, J.; Navarro-Arribas, G. Analysis of the SegWit adoption in Bitcoin. In Proceedings of the XV Reunión Española Sobre Criptología y Seguridad de la Información, Granada, Spain, 3–5 October 2018; Garcia Teodoro, P., Barragán Gil, N.M., Fuentes Garcia, R., Eds.; Universidad de Granada: Granada, Spain, 2018; pp. 230–233.
41. Kedziora, M.; Pieprzka, D.; Jozwiak, I.; Liu, Y.; Song, H. Analysis of segregated witness implementation for increasing efficiency and security of the Bitcoin cryptocurrency. *J. Inf. Telecommun.* **2023**, *7*, 44–55. [CrossRef]
42. Stütz, R.; Stockinger, J.; Moreno-Sanchez, P.; Haslhofer, B.; Maffei, M. Adoption and Actual Privacy of Decentralized CoinJoin Implementations in Bitcoin. In Proceedings of the 4th ACM Conference on Advances in Financial Technologies, Cambridge, MA, USA, 19–21 September 2022. [CrossRef]

43. Swambo, J.; Hommel, S.; McElrath, B.; Bishop, B. Custody Protocols Using Bitcoin Vaults. *arXiv* **2020**, arXiv:2005.11776.
44. Melo, D.; Hernandez, S.; Rodriguez, L.; Perez-Sansalvador, J. Bitcoin Transactions Types and Their Impact on Storage Scalability. In Proceedings of the 2023 IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Paris, France, 14–16 December 2023; pp. 1–6. [CrossRef]