An Out-Scaling Blockchain for Value Transfer with Spontaneous Sharding

Zhijie Ren and Zekeriya Erkin

Delft University of Technology,
Mekelweg 5, 2628CD, Delft, the Netherlands
{z.ren, z.erkin}@tudelft.nl,

Abstract. Blockchain technology, sometimes known by its applications like distributed ledgers and cryptocurrencies, suffers from the unscalable throughput of Byzantine fault tolerance consensus algorithms. Recently, many blockchains have been proposed to achieve out-scaling throughput, i.e., the throughput of the system grows with the number of nodes. In this paper, we propose a novel out-scaling blockchain system for the most considered type of ledger, we call Value Transferring Ledgers, in which a transaction is a transfer of some value from one node to another like Bitcoin. In our system, nodes commonly agree on a main chain and individually generate their own chains. We propose a locally executable validation scheme with uncompromised validity and scalable throughput. Furthermore, a smart transacting algorithm is introduced so that the system is spontaneously sharded for individual transactions and achieves an out-scaling throughput.

Keywords: Blockchain, distributed ledger, Byzantine fault tolerance, out-scale, off-chain

1 Introduction

Blockchain technology, made popular by Bitcoin [1], can be described as a distributed append-only database maintained by a group of untrusted nodes instead of some central authority. One of the most well-known applications of blockchain technology is cryptocurrency, in which the blockchain is in the form of a distributed ledger, i.e., the data is in the fashion of transactions which are records of value exchanges between nodes. A crucial part of the blockchain technology is the consensus algorithm that keeps the consistency on the ledger of valid transactions. The consensus algorithm can be divided into two categories, the Nakamoto-like consensus algorithms such as Proof-of-Work (POW) [1] or Proof-of-Stake (POS) [2, 3] and Byzantine fault tolerance (BFT) consensus algorithms such as PBFT [4]. For distributed ledger type of blockchain, all consensus algorithms effectively achieve the following conditions on the validity of transactions in an asynchronous network.

– **Agreement(Consistency)**: Two honest nodes should not have disagreement on the validity of a transaction.

- **Validity(Correctness)**: Invalid transactions cannot be validated by honest nodes.
- **Termination(Liveness):** All transactions will be eventually known by all honest nodes.

The term "effectively" infers that strictly speaking, the above properties are not achievable in asynchronous networks [5, 6]. However, with some additional assumptions like weak synchronous as implicitly suggested in [4], the above properties (or some weaker properties like probabilistic termination [7, 8]) can be achieved. However, blockchains with either Nakamoto-like consensus and BFT consensus are limited in scalability, i.e., the communication cost per transaction (CCPT) is lower bounded by $O(N)$.

For Nakamoto-like consensus, the POW based scheme in Bitcoin reduces the communication cost to $O(N)$ by introducing rational assumptions, incentives, and punishments on malicious behaviors. However, this scheme and its variations has limitation in the transaction rate to meet the synchronous requirements [9]. Separation of the block and transactions from leader selection would remove the dependency of the duration of the consensus process on the transaction rate, which results in $O(N)$ CCPT [3, 10, 11].

On the other hands, BFT algorithms relies heavily on the communication between nodes and have a message complexity of $O(N^2)$ for traditional algorithms like [7, 8, 4]. This results in a severe performance degradation as the population of the network grows to above two digits [9]. Recent scalable BFT algorithms such as [12–15] reduces the CCPT to $O(N)$ either by packing up transactions into a packet or for some specific scenarios.

1.1 Out-scaling Blockchains Solutions

The blockchains with the CCPT of $O(N)$ are commonly referred as "scalable blockchains", which suggests that the throughput will not increase with the number of nodes and the computation and communication capacities in the network. Recently, several solutions have been proposed to achieve $O(1)$ CCPT, sometimes referred as "out-scaling' throughput, by reducing the number of validators and recorders for each transaction. Here, we introduce three types of such schemes.

- **Off-chain Solutions:** This type of approaches mostly associated with some existing blockchain systems as the main chain. Each node holds their transactions locally, sometimes referred as "off-chain", and only sends a description or the eventual outcome of these transactions to the main chain, referred as "on-chain". Since there is no guarantee on the "off-chain" transactions, either validation nodes are introduced to validate and endorse these transactions [16, 17] or economical deposit should be provided for the transactions [18, 19]. In both cases, the Validity condition is compromised due to the centralization or the economical constraint.
- **Directed Acyclic Graph (DAG) Solutions:** In another type of approaches, we call DAG solutions, the transactions are not structured in a

chain, but in a DAG [20, 21]. The Validity condition is compromised since the validity of the transaction is not absolute. Instead, the validity is dependent on the outgoing edges of the transaction, which represents the nodes that have validated it.
- **Sharding Solutions** Recently, sharding solutions, which artificially divide the network, have been widely studied and discussed [22–25]. It includes a scheme that fairly and randomly divide the network into small shards with vanishing probability of any shard having an overwhelming number of adversaries. Hence, the BFT consensus algorithms is run only within the shards and the CCPT is then $O(g)$ where g is the size of the shard. However, the Validity condition is also compromised in the sense that the out-scaling sharding is only feasible when the ratio of adversaries in the network is small.

All aforementioned solutions has the potential to out-scale since the CCPT could be effectively reduced from $O(N)$ to $O(g)$, where g is the average number of nodes that know and validate a transaction.

1.2 Our Contributions

The main contributions of this paper are the following.

- We introduce the Value Transferring Ledgers (VTL) which brings in some realistic interpretation of the ledger. We make use of the properties that distinct value transferring transactions from generic data. First of all, a value transferring transaction can be valid or invalid and a valid transaction has some proof which could be used to prove its validity. Then, the receiver of an unspent transaction is the current owner of the transferred value. He is then motivated to show the proof of the validity of this transaction to other nodes, since he would like to spent this value.
- We proposed a novel blockchain system which consists of an existing blockchain as a main chain, individual chain for each node for its own transactions, and a locally executable validation scheme which achieves uncompromised Agreement and Validity conditions.
- For any transaction pattern, the CCPT of our system is upper bounded by $O(N)$, which suggests scalable throughput. We further introduce a smart transacting algorithm that allows our system to out-scale, i.e., reduce the CCPT to $O(1)$.

1.3 Content of This Paper

In this paper, we propose a novel out-scaling blockchain system for VTL. In Section 2, we introduce our model, design, and all important primitives. In Section 3, we introduce our validation scheme and prove the correctness of it. We analyze the performance of our scheme and introduce the concept of spontaneous sharding which results in out-scaling throughput in Section 4. In Section 5, we conclude our paper with possible topics for further exploration.

2 Model and System Description

Our system has an off-chain structure, in which the hashes of a batch of transactions, rather than individual transactions, are recorded on the main chain. In this paper, we will emphasize on our novelties and contributions. Hence, some other elements are simplified to the most comprehensive level, e.g.,

- We consider a simple value transferring system where every node holds some initial value. No incentive mechanism is included.
- The transactions has one sender and one receiver.
- We consider a Byzantine network with weak asynchronous assumption and less than $\lfloor \frac{N}{3} \rfloor$ adversaries just as [4], so that PBFT can be straightforwardly applied.
- The concept of the hashes and the digital signatures are introduced as unbreakable primitives.

In this section, we will introduce the model, the design of our system, as well as the VTL. Furthermore, we will give definitions for some terms that will be used throughout the paper.

2.1 Network Model

We consider a weak asynchronous network of N nodes connected with a Peer-to-Peer infrastructure, in which the message delay of any honest node is upper bounded by t. Each node holds some initial value that could be transacted with the others. We assume that there are $f \leq \lfloor \frac{N-1}{3} \rfloor$ Byzantine adversaries, i.e., we use the classical BFT definitions for honest nodes and adversaries.

Definition 1 (Honest nodes and Adversaries). Honest nodes *will follow the rules and only make valid transactions, which are the transactions that they can (eventually) validated.* **Adversaries** *can behave arbitrarily, including behave like honest nodes.*

The network is assumed to be permissioned, i.e., the nodes are known to each other by their identity numbers $n \in \{1, 2, \ldots, N\}$. We also assume that there exists a public key infrastructure (PKI) and nodes can link between the identity number and public key of each node. Besides, we assume that there exists unbreakable hash function $Y = H(X)$ and digital signatures $Y = Sig_i(X)$. Moreover, we introduce the "chain" as a data structure that are consists of blocks that chained with hash function. Each block consists of multiple messages and a hash digest of the previous block, except for the first block, namely the genesis block.

2.2 System Structure

Transactions We use a similar transaction based structure as the unspent-transaction-as-output (UTXO) structure described in [1] and give the following definitions for transaction and unspent transaction.

Definition 2 (Transaction). *A transaction tx_i is a five-tuple: $tx_i = \langle \text{Source}_i, s_i, d_i, a_i, r_i \rangle$ where Source_i is the set of transactions which are used as the source, s_i is the sender, d_i is the receiver, a_i is the transacted value, and r_i is the remaining value.*

Definition 3 (Unspent Transaction). *A transaction tx_i is an unspent transaction if there is no transaction tx_j which is valid and $tx_i \in \text{Source}_j$. We will explain the term "valid" in Subsection 2.3.*

Off-chain Part Each node has its own blockchain, namely individual chain, which consist of transactions which are sent by it. We define an individual chain as an ordered set $\{B_{u,1}, B_{u,2}, \dots, \}$ and a block as an ordered set $B_{u,k} = \{H(B_{u,k-1}), t_{u,k,1}, t_{u,k,2}, \dots\}$, where $t_{u,k,l}$ is a transaction. The first blocks of the chains $B_{u,1}$ are called genesis blocks. In our system we assume that there is an initial value assigned to each node. The initial value is assigned in the same fashion as a transaction, with no source. The sender and receiver of this transaction are both the node itself.

The size of a block can be arbitrary. Periodically, nodes send *Abstracts* to the *main chain* (will be introduced in the next subsection). The abstract $A_{u,k}$ is defined as the following.

Definition 4 (Abstract). *An abstract $A_{u,k}$ is a four-tuple: $A_{u,k} = \langle u, k, H(B_{u,k}), Sig_u(u|k|H(B_{u,k})) \rangle$.*

On-chain Part We consider a blockchain with PBFT as its consensus algorithm in which blocks consist of *Abstracts* signed by the corresponding nodes. This chain is called the *main chain*. We assume that the abstracts of all genesis blocks are in the main chain. Since it has been proved that the PBFT can reach BFT consensus on messages in our network [4], we simply see the main chain as a reliable and secure primitive in our system and all abstracts included on the main chain reaching BFT consensus.

2.3 Confirmation, Validity, and Proof

Apparently, the transactions on individual chains are arbitrary with neither tamper-proof nor prevention from double spending. The transactions will be tamper-proof if an abstract of a block that comes after it is contained in the main chain. We call them confirmed transactions. Here, we give the formal definition of a confirmed transaction and a confirmed block.

Definition 5 (Confirmation). *A block $B_{u,k}$ is confirmed if an abstract of this block or a block after this one, i.e., $A_{u,k'}, k' \geq k$ and all previous abstracts of the node u, i.e., $A_{u,k''}, k'' < k$, are in the main chain. A transaction $tx_i = \langle \text{Source}_i, s_i, d_i, a_i, r_i \rangle$ is a confirmed transaction of node u if $tx_i \in B_{u,k}, s_i = u$ and $B_{u,k}$ is confirmed.*

The confirmation of a transaction suggests that it is tamper-proof as if it is on-chain and it is signed.

Property 1 (Confirmed Transactions). If $t_{u,k,t} = tx_i$ is a transaction of node u in the block $B_{u,k}$ that confirmed by abstract $A_{u,k'}, k' \geq k$, then there does not exist a valid and confirmed chain $\{B'_{u,1}, B'_{u,2}, \ldots, B'_{u,k'}\}$ such that $t_{u,k,t} \neq tx_i$.

The formal proof of this property will be given in Appendix A. By Property 1, when a transaction is confirmed, the position and content of it cannot be changed. Then, we define the validity of a transaction in a traditional fashion as classical distributed ledgers.

Definition 6 (Validity of a Transaction). *A transaction* $tx_i = \langle \text{Source}_i, s_i, d_i, a_i, r_i \rangle$ *is valid if and only if the following conditions hold.*

- **Confirmed***: tx_i is confirmed.*
- **Valid source***: All transactions in $tx_j \in \text{Source}_i$ are valid.*
- **Value equality***: The original value equals to the sum of the transacted value and the remaining value, i.e., $\sum_{tx_j \in \text{Source}_i} r_j = a_i + r_i$.*
- **No double spending**[1]*: For any $tx_j \in \text{Source}_i$. If there exists a valid transaction tx_l that $tx_j \in \text{Source}_l$, then $tx_l = tx_i$.*

We then give the definitions of a validation function and the validity proof of a transaction.

Definition 7 (Validation Function, Validity Proof, and Validation Scheme). *Given a function we called* validity proof $y = f(x)$ *and a function we called* validation function $y = g(x, f(x))$*, if* $g(x, f(x)) = \mathsf{valid}$ *if and only if tx_i is valid, we call $y = f(x)$ and $y = g(x, f(x))$ form a* validation scheme.

2.4 Value Transferring Ledgers

We introduce a special type of ledgers in which the transactions are records of value transfer. The concept itself is rather comprehensive and realistic. For instance, in cryptocurrencies, the transactions are records of the transferring of the currency. The main difference of VTL from other types of databases is that in VTL, every transaction is a transfer of positive value, which suggests the following. Firstly, the receiver of a transaction is the holder of that value until it is spent again. In this period, this value can be considered as his property and he should take full initiative and responsibility of proving the existence and the authenticity of the value. If he fails to do so, it will be consider as against his own interest. For instance, if the value is considered as money, the holder of the money is motivated to prove the money is real when he uses it to purchase.

[1] We define this property exclusively for a block, i.e., if there are multiple transactions using one source in a block, instead of letting the first one to be valid and invalidated the rest, we invalidate all of them. Note that if there are multiple transactions using one source in different blocks, then the first one is valid and the rest are invalid.

A failure in proving will cause the purchase to fail, which is against his own interest.

Secondly, the concern of the nodes are validity and authenticity of the value instead of the transaction records. Hence, nodes will check the past transaction records if they are parts of the proof that supports the authenticity of the value. Otherwise, nodes will not care about the validity of a past transaction.

Here, we first define the term *curiosity* and then give formal descriptions of the properties of VTL.

Definition 8 (Curiosity). *A node u is* curious *about transaction tx_i if it would like to check the validity of tx_i.*

Property 2 (History disinterest). A node u is curious about a spent transaction tx_i only when it is curious about an unspent transaction tx_j and the validity of tx_j is necessary for the validity of tx_i.

Property 3 (Rational Receiving). A node u is curious about transaction tx_i if it is the receiver of tx_i and does not know the validity of it.

Property 4 (Rational Owner). If node u is the receiver of a valid and unspent transaction tx_i, it will provide $\mathcal{P}(tx_i)$ to any node once it is required.

In practice, it is not rational for a node to validate an unspent transaction if it not the receiver of it since validation is resource consuming. Hence, we have an alternative version for Property 3 to minimize the cost in a resource-limited network.

Property 5 (Rational and Cost-saving Receiving). A node u is curious about transaction tx_i if and only if it is the receiver of tx_i and does not know the validity of it.

This property will not effect the correctness of our scheme. It will be applied in the performance analysis for simplicity.

3 Validation Scheme

In this section, we first introduce our validation scheme by describing the validity proof and the validation function of a transaction. Then, we prove the correctness of this scheme and BFT consensus achieving of valid transactions in VTL model.

3.1 Validity Proof and Validation Function

Here we define the validity proof of transaction tx_i.

Definition 9 (Validity Proof). *Assuming that $s_i = u$ for transaction tx_i, $tx_i \in B_{u,k}$, and there exists an abstract $A_{u,k'}, k' \geq k$, a validity proof $\mathcal{P}(tx_i)$ is a set of all blocks before and including $B_{u,k'}$ union with proofs of all transactions in* Source_i*, i.e., $\mathcal{P}(tx_i) = \{B_{u,k''} | k'' \leq k'\} \cup \{B_{v,l} | B_{v,l} \in \mathcal{P}(tx_j), tx_j \in \text{Source}_i\}$.*

By Definition 9, the validity proof of tx_i of sender u includes the chain of u from the genesis block to the block which has an abstract in the main chain. Moreover, it also includes the chains of the sources of this transaction, and recursively the sources of the sources until the genesis block.

In the following lemma, we show that the proof of unspent transactions can always be collected.

Lemma 1 (Feasibility of the Proof Collection). *If a node u is curious about a valid transaction tx_i, then it can always identify a node v such that would provide the proof of it.*

Proof. If tx_i is an unspent transaction, this lemma directly follows from Property 4 since the receiver of tx_i will provide it. If tx_i is a spent transaction, then by Property 2, u will only be curious about tx_i if u is curious about an unspent transaction tx_j and the validity of tx_i is required in tx_j. Then, by Definition 9, we have $\mathcal{P}(tx_i) \subset \mathcal{P}(tx_j)$. Then, by Property 4, u can collect the proof of tx_j from the receiver of tx_j.

By Lemma 1, the proof of a transaction tx_i can always be collected via point-to-point communication. However, although by our model the receiver of the unspent transaction is motivated to provide the correct proof, the requester of the proof will not accept it as proof without his own verification. We give the Proof Verification Algorithm in Appendix B.

If the verification result is pass, it suggests that $\mathcal{P}(tx_i)$ is indeed a validity proof for transaction tx_i since the algorithm is a direct translation from the definition of the validity proof.

The validation function is defined as Algorithm 1.

Algorithm 1 Validation Function $\mathsf{V}(tx_i, \mathcal{P}(tx_i)), tx_i \in B_{u,k}$

$\langle \{txs_{i,[1:n]}\}, s, d, a, r \rangle \leftarrow tx_i$
$\langle *, *, *, *, rs_{i,[1:n]} \rangle \leftarrow txs_{i,[1,n]}$
#Proof Check
if $\mathsf{Ver}(\mathcal{P}(tx_i)) \neq \mathsf{pass}$ **then return** unknown
#Equality Check
if $\sum(rs_{i,1:n}) \neq a + r$ **then return** unknown
#Double-Spending Check
for $B_{s,m}, m = [1 : k]$ **do**
  for All transactions tx_j in $B_{s,m}$ **do**
    if $\text{Source}_j \cap \text{Source}_i \neq \emptyset$ **then return** unknown
#Source Check
for $txs[d], d = 1 : n$ **do**
  if $\mathsf{V}(txs[d], \mathcal{P}(txs[d])) \neq \mathsf{valid}$ **then return** unknown
return valid

3.2 Correctness of the Validation Scheme

Here, we prove that the proposed validity proof and validation function forms a validation scheme as defined in Definition 7.

Theorem 1. $\mathsf{V}(tx_i, \mathcal{P}(tx_i)) = \mathsf{valid}$ *if and only if* tx_i *is valid.*

Proof. We first prove that if $\mathsf{V}(tx_i, \mathcal{P}(tx_i)) = \mathsf{valid}$ then tx_i is valid. It directly follows from the four checks in Algorithm 1 since they are exactly the conditions in Definition 6.

We then show that if $\mathsf{V}(tx_i, \mathcal{P}(tx_i)) \neq \mathsf{valid}$ then tx_i is not valid. To prove this, we first prove the statement "if $\mathsf{V}(tx_i, \mathcal{P}(tx_i)) \neq \mathsf{valid}$ and $\forall tx_j \in \text{Source}_i, \mathsf{V}(tx_j, \mathcal{P}(tx_j)) = \mathsf{valid}$, then tx_i is not valid."

We prove this statement by contradiction. Assuming that there exists a transaction tx_k such that $\mathsf{V}(tx_k, \mathcal{P}(tx_k)) \neq \mathsf{valid}$ but for all $tx_j \in \text{Source}_k, \mathsf{V}(tx_j, \mathcal{P}(tx_j)) = \mathsf{valid}$, and tx_k is valid.

By our algorithm, at least one of the four checks other than the "**Source Check**" is failed. If the step "**Proof Check**" fails, it suggests that a proof $\mathcal{P}(tx_i)$ does not exist, which contradicts the assumption that tx_i is valid. If the step "**Equality Check**" fails, it contradicts the **Value equality** condition of valid transaction. If the step "**Double-Spending Check**" fails, it contradicts the **No double spending** condition of valid transaction.

We then prove that if $\mathsf{V}(tx_i, \mathcal{P}(tx_i)) \neq \mathsf{valid}$ then tx_i is not valid by contradiction. If this does not hold, then there must exist a transaction that violates the statement proved above. This transaction might be tx_i, the source of tx_i, or recursively one in the sources of the sources.

3.3 BFT Satisfactory

In this section, we show that our system satisfies the Agreement and Validity condition of BFT as described in [12, 13] with a compromised Termination condition for all valid transactions.

Theorem 2 (BFT Satisfactory). *Our system satisfies the following conditions in VTL model. Here, we use the term "node u validate a transaction* tx_i*" to represent that node u runs a validation function on* tx_i *with the result* valid*.*

- **Agreement(Consistency)***: If an honest node validated a transaction, then, if another honest node is curious about this transaction, it will also validate it.*
- **Validity(Correctness)***: If an honest node proposed a transaction and at least one honest node is curious about it, this transaction will be validated by at least one honest node.*
- **Termination(Liveness):** *If a transaction is proposed by an honest node, then all honest nodes that are curious about it can collect the proof of it.*

Proof.

- **Agreement:** If a transaction tx_i is validated by an honest node, i.e., $\mathsf{V}(tx_i, \mathcal{P}(tx_i)) = \mathsf{valid}$, by Theorem 1, tx_i is valid. Then, by Property 1, if another node is curious about tx_i, the proof can be collected. Then, since the validation function is deterministic, another curious honest node will also run the validation function that result in valid.
- **Validity:** If a transaction is proposed by an honest node, by the definition of the honest node, the node can eventually validate it and we have $\mathsf{V}(tx_i, \mathcal{P}(tx_i)) = \mathsf{valid}$. Then, by Lemma 1, the proof of this transaction can be collected. Then, since the validation function is deterministic, the honest node that is curious about it will validate it.
- **Termination:** If a transaction is proposed by an honest node, by the definition of honest nodes this transaction will eventually be confirmed and can be validated by itself. Then, by Property 1, any nodes that is curious about it can collect the proof of it.

The insight of Theorem 2 is the following. Firstly, we explicitly focus on achieving BFT consensus for valid transactions. Then, for each valid transaction, we introduce the curious nodes based on Property 2 of the VTL model, which sets a constraint on the nodes that would like to know and validate the transaction and thus compromises the Termination condition. However, we prove that the Agreement and Validity conditions are not compromised.

4 Performance Analysis and Spontaneous Sharding

In this section, we will give theoretical explanations for the out-scale claim that we made for the throughput, i.e., the CCPT could be reduced to $O(1)$. First we show that the CCPT is simply the size of the proof of individual transactions, which is shown to be highly dependent on the transaction pattern of the network. We compare the CCPT to classical blockchain systems and show that the throughput of our system is scalable and can straightforwardly out-scales for some specific transaction patterns. Then, we propose a simple smart transacting algorithm which could result in spontaneous sharding of the network for each transaction and out-scaling throughput. Simulative results are given to support our claim.

4.1 Communication Cost Per Transactions

In our system, the main chain is using PBFT with $O(N^2)$ message complexity. However, the sizes of the individual chains are arbitrary and independent of the main chain. As a result, the communication cost of the main chain becomes a negligible term in CCPT. The duration of the consensus process still plays an important role in the latency of our system. However, note that the PBFT-based scheme is used only for easy comprehension and can be easily replaced by other scalable and low latency blockchain systems to improve the latency of our system.

Then, in our system, the CCPT is in fact the cost in collecting the proof of that transaction. We assume that rational nodes will not care about invalid transactions and thus will not try to re-collect the proof of a transaction if it is failed for a number of times. In other words, malicious nodes cannot spam invalid proofs to jam the network. In this case, the CCPT of our system can be represented by $O(hp)$, where h is the average number of nodes that is curious about a transactions before it is spent (once it is spent, the proof is included in the proof of another unspent transaction) and p is the average size of the proofs. For the simplicity in analysis, we consider a resource limited network with Property 5. In that case, the CCPT of our system is $O(p)$.

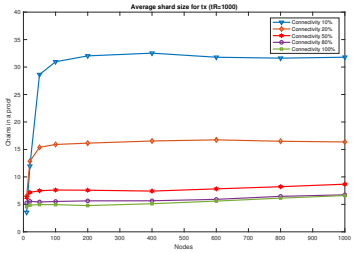
Now we focus on the size of the proof. In general, the proof of a transaction tx_i includes the chains of the sender, the senders of all sources of this transaction, and the senders of recursively the sources of the sources, which is a lot at the first sight. However, it is actually lightweight comparing to all classical blockchain systems. For example, the validation of any transaction in Bitcoin requires an exhaust validation on all transactions before it to prevent double-spending. However, in practice, the storage is traded for validation efficiency so that the validated transactions are stored and the validation is done incrementally by the miners. The same holds for our system. In a setting where the storage is not limited, in the worst case all nodes need to be updated with all transactions. This can be done in a point-to-point and reliable fashion with $O(N)$ CCPT since by Property 4 and Lemma 1, there is no need for BFT agreement scheme to guarantee the reliability of the proofs. On the contrary to the worst case, a better case would be that the transaction pattern is separated into fractions and the nodes only make intra-fraction transactions. In that case, proof of any transaction contains the chains of only the nodes in their fractions and the proof size of a transaction is $O(g)$, where g is the size of the fraction. As a result, our system achieves out-scaling throughput.

4.2 Spontaneous Sharding

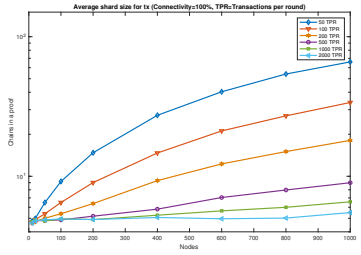
Here, we consider a more general case that the transaction pattern is not separated into small fractions and show that our system could still achieves out-scaling throughput if all nodes behaves rationally. We call this spontaneous sharding. The idea behind the spontaneous sharding is simple: rational nodes will try to minimize their transmission costs by minimizing the proof size of each transaction. Then, according to the information about the chains that the receiver already has, the sender will choose from all its unspent transactions for the ones with the least amount of required proofs. For example, if node 1 has transacted with a receiver node 2 before with proofs including the chains from nodes $\{3, 4, 5, 6\}$. Then, for all future transactions that it makes to node 2, it will always use the unspent transactions with proofs that consist of the chains from $\{3, 4, 5, 6\}$ so that it only needs to update these chains. In other words, if all nodes are rational, the value of a transaction will tends to only cycle in a group of nodes instead of the whole network. As a result, for each transaction, the network is spontaneously sharded.

We give a smart transacting algorithm in Appendix C for rational nodes, which simply let the sender detect the chains that the receiver already has and choose the sources accordingly. The detection can be both non-interactive and interactive. The former is based on the knowledge of the sender on the transaction patterns and the latter can be done with the receiver simply telling the sender about the chains that it already has.

Here we show some simulative results of spontaneous sharding. We assume that each node only transacts with c nodes in the network with equal probability. The transacted amount is a fraction of his money chosen uniformly at random. The confirmation time and the transaction rate are assumed to be t and R, respectively. Hence, for each transaction, a node will minimize the proof size of the transactions in this round by deliberately choosing from tR transactions confirmed by the last on-chain abstract. We consider a stable state with sufficient transactions been made by each node. In this case, either interactive or non-interactive schemes will have the same result since the transaction pattern is fixed and each node should already have enough prior knowledge for the chains that each receiver has. In the simulation, we focus on the spontaneous sharding, i.e., the average number of chains that is required as proof for each transaction.



(a) Fix ratio of connectivity.



(b) Full connectivity with variant transactions per round (TPR).

In Fig 1(a), it is shown that if the ratio of the connectivity c/N is fixed, the number of chains included in the proof of a transaction tends to be a constant when the network grows large.This result is reasonable since in the extreme case that all nodes transact with everybody else, if node A wants to transact with node B, it can always find an unspent transaction sent by node B if it waits long enough. In that case, the proof size is minimized to two chains only. This relationship is shown in Fig. 1(b) that the number of chains in proofs will decrease as the number of transactions per round increases.

5 Conclusion and Future Work

In this paper, we proposed a novel blockchain system for the most considered type of distributed ledgers which we called VTL (cryptocurrency falls into this

category). Our system has a very simple and fully decentralized structure that does not introduce any node serving as "leader" or "validator". Our system achieves uncompromised Agreement and Validity conditions on the valid transactions and has a potential to out-scale by spontaneous sharding.

For future work, we believe the following topics are interesting for further exploration.

- **Supportive to conditional payments/smart contracts:** We conjecture that conditional payments and smart contracts can also by supported by this system with modified data structure and validation scheme as long as each transaction includes some value transferred to at least one of the receiver.
- **Non-VTL model:** Our system exploits the features of VTL and thus only works if the receiver has interest in the transaction. Otherwise the receivers will not be motivated to provide the proof of the transaction and other nodes will not be able to know this transaction unless they contact the senders. How to use a similar system for non-VTL models remains a non-trivial problem.
- **Total ordering:** The consensus of individual transactions in our system is explicitly put on the validity of the transactions. The order of the transactions is determined by the appearances of their corresponding abstracts on the main chain. Hence, the transactions that confirmed by the same block in the main chain is considered as simultaneous. The consensus of the ordering of the transactions remains an interesting problem, especially for some applications that requires total ordering of transactions.

References

1. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. (2008)
2. Sunny King, S.N.: Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. (2012) Available at `https://peercoin.net/assets/paper/peercoin-paper.pdf`.
3. Micali, S.: ALGORAND: the efficient and democratic ledger. CoRR **abs/1607.01341** (2016)
4. Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OSDI. Volume 99. (1999) 173–186
5. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2) (April 1985) 374–382
6. Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News **33**(2) (June 2002) 51–59
7. Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience. In: Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing, ACM (1994) 183–192
8. Bracha, G.: Asynchronous byzantine agreement protocols. Information and Computation **75**(2) (1987) 130–143
9. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E.G., et al.: On scaling decentralized blockchains. In: International Conference on Financial Cryptography and Data Security, Springer (2016) 106–125

10. Eyal, I., Gencer, A.E., Sirer, E.G., Van Renesse, R.: Bitcoin-NG: A scalable blockchain protocol. In: 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), USENIX Association (2016) 45–59
11. Pass, R., Shi, E.: Hybrid consensus: Efficient consensus in the permissionless model (2016)
12. Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: Reliable Distributed Systems, 2005. SRDS 2005. 24th IEEE Symposium on, IEEE (2005) 191–201
13. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of BFT protocols. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM (2016) 31–42
14. Guerraoui, R., Knežević, N., Quéma, V., Vukolić, M.: The next 700 BFT protocols. In: Proceedings of the 5th European conference on Computer systems, ACM (2010) 363–376
15. Crain, T., Gramoli, V., Larrea, M., Raynal, M.: (leader/randomization/signature)-free byzantine consensus for consortium blockchains. CoRR **abs/1702.03068** (2017)
16. Lerner, S.D.: Rsk: Bitcoin powered smart contracts. (2015) Available at `https://uploads.strikinglycdn.com/files/90847694-70f0-4668-ba7f-dd0c6b0b00a1/RootstockWhitePaperv9-Overview.pdf`.
17. WOOD, G.: Polkadot: Vision for a heterogeneous multi-chain framework. (2016) Available at `http://www.the-blockchain.com/docs/Gavin%20Wood%20-%20Polkadot%20-%20%20Vision%20For%20A%20Heterogeneous%20Multi-chain%20Framework.pdf`.
18. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments. Technical Report (draft) (2015) Available at `https://lightning.network/lightning-network-paper.pdf`.
19. Poon, J., Buterin, V.: Plasma: Scalable autonomous smart contracts. (2017) Available at `https://plasma.io/plasma.pdf`.
20. Popov, S.: The tangle. (2014)
21. Churyumov, A.: Byteball: A decentralized system for storage and transfer of value. (2016) Available at `https://byteball.org/Byteball.pdf`.
22. Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Ford, B.: Omniledger: A secure, scale-out, decentralized ledger. IACR Cryptology ePrint Archive
23. 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, New York, NY, USA, ACM (2016) 17–30
24. Buterin, V.: On sharding blockchains. Sharding FAQ (2017) Available at `https://github.com/ethereum/wiki/wiki/Sharding-FAQ`.
25. Al-Bassam, M., Sonnino, A., Bano, S., Hrycyszyn, D., Danezis, G.: Chainspace: A sharded smart contracts platform. CoRR **abs/1708.03778** (2017)

A Proof of Property 1

Proof. We proof this property by contradiction. If there exists a chain $\{B'_{u,1}, B'_{u,2}, \ldots, B'_{u,k'}\}$ which is confirmed and valid such that $t'_{u,k,t} \neq tx_i$. First of all, a valid chain suggests that all hashes are correct and a confirmed chain suggests that all abstracts of $\{B'_{u,1}, B'_{u,2}, \ldots, B'_{u,k'}\}$ is on the main chain. Then,

since $t_{u,k,t} = tx_i$ is confirmed by an abstract $A_{u,k}$, all abstract of the chain $\{B_{u,1}, B_{u,2}, \ldots, B_{u,k'}\}$ are also on the main chain. Moreover, since the abstracts are signed and the digital signatures are assumed to be unbreakable, the chains $\{B'_{u,1}, B'_{u,2}, \ldots, B'_{u,k'}\}$ and $\{B_{u,1}, B_{u,2}, \ldots, B_{u,k'}\}$ will have the same set of abstracts of node u on the main chain, which includes $A_{u,k'}$ as the abstract of both $B'_{u,k'}$ and $B_{u,k'}$. Since the hash function is assumed to be unbreakable, we have $B'_{u,k'} = B_{u,k'}$. Then, if $t'_{u,k,t} \neq tx_i$, then we will have $B'_{u,k} \neq B_{u,k}$ and $B'_{u,k'} = B_{u,k'}, k' \geq k$, which contradicts our assumption of the unbreakable hash function.

B Proof Verification Algorithm

Algorithm 2 Proof Verification Algorithm $\mathsf{Ver}(\mathcal{P}(tx_i)), tx_i \in B_{u,k}$

```
#Verify the chain including this transaction
count ← 0
absmark ← 0
txs[1 : n] ← Source_i
s ← s_i
for B_{s,m}, m = 1 : max do                          ▷ Check the integrity of the chain
    if tx_i ∈ B_{s,m} then count + +
    if m ≠ 1 and  first element in B_{s,m} ≠ H(B_{s,m−1}) then return fail
    if A_{s,m} is included in the main chain then
        absmark ← m
        if H(B_{s,m}) ∉ A_{s,m} then return fail
if absmark < k then return fail                      ▷ Check the confirmation
if count ≠ 1 then return fail                        ▷ Check the existence of the transaction
#Verify the chains of the sources
for txs[d], d = 1 : n do
    if Ver(txs[d]) ≠ pass then return fail
return pass
```

C Smart Transacting Algorithms

Here we give a smart transacting algorithms $\text{Source}_i = \mathsf{ST}(d_i, a_i, \mathcal{C}_u)$ in Algorithm 3 for rational nodes, where node u intends to send an amount of a_i to node d_i in transaction tx_i and $\mathcal{C}_u$ is a collection of all transactions and proofs recorded in node u.

Note that Algorithm 3 is a non-interactive algorithm. The choice of the sources is much easier in an interactive fashion, in which the receiver simply tells the sender the chains that he has for the second step of Algorithm 3. Both interactive and non-interactive algorithms will result in spontaneous sharding.

Algorithm 3 Non-interactive Smart Transacting Algorithm $\mathsf{Source}_i = \mathsf{ST}(d_i, a_i, \mathcal{C}_u)$

#Step 1: Check for all unspent transactions
$\mathsf{UT} \leftarrow$ all unspent tx_i that are in $\mathcal{C}$.
#Step 2: Determine the chains that d already has according to $\mathcal{C}$
$\mathsf{Collected} \leftarrow \emptyset$
for each tx_i in $\mathcal{C}$ **and** $d_i = d$ **do**
    $\mathsf{chains}_i \leftarrow \{v | \mathcal{B}_v \in \mathcal{P}(tx_i) \cap \mathcal{C}_u\}$ ▷ All chains in the proof of tx_i according to $\mathcal{C}_u$
    $\mathsf{Collected} \leftarrow \mathsf{Collected} \cup \mathsf{chains}(i)$
#Step 3: Find the sources which has the least amount of chains to send
for all $\mathsf{Source}_n \subset \mathsf{UT}$ such that the sum amount no less than a_i **do**
    $\mathsf{Proof}_n \leftarrow$ union of all $\mathcal{P}(tx_i), tx_i \in \mathsf{Source}_n$
    $\mathsf{NChains}_n \leftarrow \{v | \mathcal{B}_v \in \mathsf{Proof}_n\}$
    $\mathsf{ToCollect}_n \leftarrow \mathsf{NChains}_n / \mathsf{Collected}$
return Source_l where $\mathsf{ToCollect}_l = \min(|\mathsf{ToCollect}_n|)$