Sharding in permissionless systems in presence of an adaptive adversary

Emmanuelle Anceaume, Davide Frey, Arthur Rauch

Sharding in permissionless systems in presence of an adaptive adversary

Emmanuelle Anceaume, Davide Frey, and Arthur Rauch

IRISA/ Université de Rennes/ CNRS/ Inria, France

Abstract. We present SplitChain, a protocol intended to support the creation of scalable proof-of-stake and account-based blockchains without undermining decentralization and security. This is achieved by using sharding, i.e. by splitting the blockchain into several lighter chains managed by their own disjoint sets of validators called shards. These shards balance the load by processing disjoint sets of transactions in parallel. SplitChain distinguishes itself from other sharded blockchains by reducing the synchronization constraints among shards while maintaining security guarantees in an asynchronous setting. A dedicated routing protocol enables transactions to be redirected between shards with a low number of hops and messages. Finally, the protocol is designed to dynamically adapt the number of shards to the system load to avoid over-dimensioning issues encountered in static sharding-based solutions.

Keywords: Blockchain, Sharding, Distributed ledger, Scalability

1 Introduction

Blockchain technology is well known to provide a tamper-proof "append-only" distributed-ledger abstraction. The immutable nature of this ledger implies a constant growth of its storage requirements, with every block being retained for eternity. Not only does this impact storage, but it also increases the communication requirements for a new honest party to join the system as it needs to download the entire blockchain from the network in order to have a consistent view of the transaction history. Sharding is a scaling solution involving splitting the blockchain into several smaller blockchains, called "shards", each processed and stored by its respective set of validators. Allocating validators to separate shards better balances communication and storage loads, accommodating more efficient network growth to achieve near-linear throughput scalability as the number of shards increases. However, sharding poses several challenges.

Firstly, since the system is partitioned, a shard can be compromised using a smaller malicious proportion of the total number of validators compared to what is required for classic blockchains. In order to ensure shard safety, it is necessary to implement an unbiased and verifiable random allocation of validators to prevent the adversary from targeting a particular shard. It is also necessary to periodically relocate validators to prevent the adversary from amassing corrupted validators inside a shard. Secondly, it is necessary to ensure both the

verification and atomicity of cross-shard transactions. A cross-shard transaction refers to a transaction made between users managed by different shards. As each shard only knows the state of the users stored in its blockchain, it is necessary to ensure that *(i)* an invalid transaction will not be accepted by any shard, and *(ii)* a valid transaction partially accepted by the involved shards can be aborted so that funds cannot be locked up indefinitely or duplicated. To solve both problems there must exist some minimal amount of synchronization among shards. Most sharded-based solutions, including Elastico [**?**], Omniledger [**?**], RapidChain [**?**] or Ethereum 2.0, achieve this using a global synchronization blockchain maintained by every validator in addition to their shard's blockchain. In addition, sharding solutions based on a fixed number of shards (static sharding) are penalized by an uneven distribution of transactions, as revealed by our experiments on Ethereum.

Contributions. We propose SplitChain, a fully decentralized state sharding solution such that,

- Shards progress at their own pace without requiring the maintenance of a synchronization blockchain or any heavy synchronization mechanisms;
- Shards keep a loosely synchronized view of each other's state to guarantee the processing of any cross-shard state updates in a bounded number of consensus executions;
- Shards are tolerant to a fast adaptive adversary controlling less than a third of all validators by relying on a novel distributed attribution mechanism;
- Shards self-adapt to the current payload of the system by self merging and splitting the set of accounts, while adapting to the presence of hot-spots;
- Shards efficiently forward transactions by leveraging the properties of hypercubic routing protocols.

The remainder of the paper is as follows: Section **??** presents the system model. Section **??** describes a chain's structure and block creation. Section **??** is dedicated to the management of transactions that span different chains and the assignment of validators to chains. Section **??** introduces the system's routing protocol. Section **??** investigates SplitChain's security properties. Finally, Section **??** compares SplitChain to previous state-of-the-art sharding solutions.

2 Model of the system

2.1 Accounts and validators.

Splitchain uses the account model to provide durable identities to its users and to enforce single-input single-output transactions for easier transaction management. Accounts are divided into two categories: user accounts and validator accounts. Users can send and receive transactions, while validators participate in SplitChain's protocol.

Definition 1 (Accounts). *Accounts are persistent balances identified by a public key hash.*

Definition 2 (Validators). *Validator accounts are special accounts created by users to participate in SplitChain's protocol. All validator accounts possess the same constant amount of currency called stake.*

Any user can create a validator by submitting the corresponding amount of stake through a transaction. Users may possess multiple validators concurrently. Stake serves as a limited resource to render the cost of Sybil attacks impractical. Users specify a (bounded) number of "*cue*" blocks (see Definition **??**) for the existence of their validator, after which the validators expires and its stake is returned.

2.2 Network model.

We consider a peer-to-peer network of validators that self-divide into several chains. Both the number of chains and validators vary over time to withstand a dynamic and open environment. Communication delays are finite but not bounded, in particular there are no bounds on the time taken to deliver messages or any preservation of the order of those messages, conforming to the asynchronous communication model.

2.3 Threat model.

Definition 3 ((μ,δ)-adaptive adversary). *We consider a Byzantine adversary that never controls more than a fraction $0 \leq \mu < 1/3$ of validators.*[1] *Furthermore, the adversary is adaptive over a period $\delta \geq 3$, in the sense that if the adversary chooses to corrupt some newly attributed validator, this corruption will be effective after δ successive consensus executions.*

2.4 Cryptographic functions.

Beyond common cryptographic functions, i.e. a hash function and asymmetric signatures, validators use verifiable random functions(VRF) [**?**] to generate pseudo-random hashes. VRFs are computed privately using a validator's private key, however their result is publicly verifiable using the validator's public key. Validators also use Merkle trees to prove the inclusion of data in a dataset without knowing its content. Every leaf of the Merkle tree is labelled with the hash of a data item, and every node other than the leaves is labelled with the hash of the concatenation of the labels of its child nodes. The root of the tree serves as the commitment to the dataset, and the Merkle path of a data, i.e. the hashes of the sibling nodes of the nodes that connect a leaf to the root of the Merkle tree, serves as the proof of the inclusion of the data. The (μ, δ)-adversary's computational power is restricted to match standard cryptographic assumptions.

[1] Given that validators all hold the same amount of stake, this equates to controlling less than μ of the total validation stake.

3 Structure of the blocks

There are two types of blocks in SplitChain: *chaining blocks* and *transaction blocks*. Each transaction block is paired with a chaining block: the two constitute the outcome of a single consensus execution. The index of a block refers to its position in the chain.

Definition 4 (Transaction blocks). *Transaction blocks contain all the transactions accepted by validators during a consensus execution.*

Definition 5 (Chaining blocks). *Chaining blocks store the hash of the previous chaining block of their chain and the Merkle root of the transaction block produced during the same consensus execution.*

Definition 6 (Cue block). *Let N_{cue} be some positive integer. $\forall k \geq 0$, every chaining block at position $k \times N_{cue}$ of a chain is called a cue block. A cue block points to both the previous chaining block and the previous cue block.*

Chaining blocks are akin to traditional block headers. A chaining block of a chain C contains a list of hashes. In particular, it provides the Merkle roots of the transaction block produced during their consensus execution, and of the accounts managed by the chain as well as the fingerprints needed for cross-chain transaction verification. It also contains the list of the latest known chaining block indices of the chains other than C, which we refer to as the fingerprint of the block. The interested reader may refer to Figure **??** in the Appendix. For synchronization purposes, chaining blocks are disseminated to all chains using SplitChain's dedicated routing protocol described in Section **??**. In contrast, transaction blocks of chain C are only stored by the validators participating in C's consensus executions.

4 Handling multiple chains

4.1 Assigning validators to chains

Each block of each chain is created via the local execution of a consensus algorithm. Consensus committees are periodically renewed (at each consensus execution) and their members are selected via two stages. The rationale of both stages is to prevent the (μ, δ)-adversary from predicting, and thus manipulating, members of consensus committees. Briefly, starting from their *initialization chains*, validators are uniformly distributed over their *reference chains* (first stage), and then uniformly distributed over their *consensus chains* (second stage). Validators stay forever in their initialization chain (that is as long as they want to actively participate in the creation of SplitChain's blocks), they stay for N_{cue} blocks (see Definition **??**) in their reference chain, and finally they stay for no more than the duration of three consensus executions in their consensus chains. Hence, for any chain C_i, and for any validators v, v' and v'' of Splitchain, at time t, C_i can be the initialization chain of v, while it is the reference chain of v' and finally the consensus chain of v''.

Initialization chain Any user u wishing to actively participate in block creation must first register a *validator account*, with a given amount of stake. Validator accounts allow users to participate in different consensus executions. A single user can create multiple validator accounts if they own enough currency. Let $v = (sk_v, pk_v)$ be u's validator. User u submits a transaction to instantiate v's stake on what we call v's *initialization chain*. This chain, denoted by $C^{\text{init}(v)}$, is the chain whose label is the closest to $addr_v = H(pk_v)$ (by closest we mean the chain whose label minimizes the numerical value of the xor between $addr_v$ and the chain's label. The routing mechanism is described in Section **??**). Initialization chains provide a stable anchor point for validators and allow them to prove the existence of their accounts. However, they do not contribute to SplitChain's security. Indeed, some malicious validator v' can iteratively invoke the cryptographic hash function to sit on some targeted chain $C^{\text{init}(v')}$. The current consensus committee of $C^{\text{init}(v)}$ updates the list L of new validators with v and inserts L's Merkle root into the chaining block under construction (see "Future Initialization Proof" in Figure **??**). When the next cue block of $C^{\text{init}(v)}$ is created (see Definition **??**), the consensus committee in charge of that cue block will assign each new validator of L to their *reference chain* (see Algorithm **??** in Appendix **??**). Specifically, Algorithm **??** shuffles L (using the seed of the cue block), partitions L into N sub-lists of $\lfloor |L|/N \rfloor$ validators, where N is the current number of chains of SplitChain, and assigns each random sub-list to one of the N chains of SplitChain. The Merkle root of L and the proof of validator assignment to their reference chain is included in the cue block (see Algorithm **??** in Appendix **??**). The chain to which v will be assigned is called v's *reference chain* and is denoted by $C^{\text{ref}(v)}$. At every cue-block creation, Algorithm **??** is executed so that v is periodically re-assigned to a new reference chain.

Reference chain Each reference chain $C_1, \ldots, C_N$ of SplitChain assigns uniformly at random its V/N referenced validators to the N *consensus chains* of SplitChain, where V is the total number of referenced validators in SplitChain. It is important to note that to face a (μ, δ)-adaptive adversary, validator v can not be attributed to the consensus committee of $C^{\text{cons}(v)}$ more than $\delta - 1$ consecutive times. Hence to be assigned to its consensus chain $C^{\text{cons}(v)}$, validator $v = (sk_v, pk_v)$ generates its consensus credential as a new key pair $(sk_{\text{cons}(v)}, pk_{\text{cons}(v)})$ and sends a signed credential storage request to the current consensus committee of $C^{\text{ref}(v)}$. Validator v regenerates its credentials every δ attributions. This signed request contains $addr_v$, ref_v and $H(pk_{\text{cons}})$, proving the legitimacy of v's request. If the request is valid, v's consensus credential is added by the current consensus committee of $C^{\text{ref}(v)}$ to the list L of validators that will be used by Algorithm **??** to build N sub-lists of $\lfloor |L|/N \rfloor = V/N^2$ validators each. Such a sub-list is called a *reference set*, and each sub-list is assigned uniformly at random to one of the N chains os SplitChain. The chain to which v is assigned is called v's consensus chain and is denoted by $C^{\text{cons}(v)}$. The Merkle root "Credentials" (see Figure **??**) of this list is added to the chaining block. Merkle paths are sent to the referenced validators of $C^{\text{ref}(v)}$ to serve as

credential proof. The current consensus committee of $C^{\text{ref}(v)}$ sends to each consensus chain a transaction that contains the size of the reference set they have allocated to it. This allows validators to know the total amount of stake inside their consensus committee.

Consensus chain To join the consensus committee of $C^{\text{cons}(v)}$, v issues a "join" request to $C^{\text{cons}(v)}$ using SplitChain's dedicated routing protocol. Upon receipt of the request, the core validators of $C^{\text{cons}(v)}$ directly reply to v with the list of core validators (see Section **??**) and a sub-list of the current consensus committee members chosen at random. We call these validators the bootstrapping validators of v. The number of bootstrapping validators is large enough so that, with high probability $1 - \varepsilon$, with $\varepsilon \in (0, 1)$, at least one of them is honest. As chaining blocks cannot be forged by the (μ, δ)-adversary without taking over a chain's consensus, the proofs they contain are sufficient to verify the authenticity of the data provided by the bootstrapping validators. Thus, only one honest bootstrapping validator is sufficient for a successful bootstrap of v to $C^{\text{cons}(v)}$. Bootstrapping validators will provide v with $C^{\text{cons}(v)}$'s state. This state corresponds to the latest state of $C^{\text{cons}(v)}$'s user accounts, the proofs of attribution of the current consensus committee and the list of core validators. Note that the fingerprint of the latest chaining block b of $C^{\text{cons}(v)}$ contains the latest chaining-block index of each chain of SplitChain known to the consensus committee of $C^{\text{cons}(v)}$ when b was created. Therefore, the latest chaining block b contains the chaining block index of each reference chain used to create the current consensus committee of $C^{\text{cons}(v)}$. From this list, v establishes the list of attribution proofs of the committee, requesting the missing chaining blocks from its bootstrapping validators if necessary. Algorithm **??** in Appendix **??** presents the detailed pseudo-code of v's attribution to its consensus chain.

4.2 Creation of the blocks of a chain

As described above, each chain C_i of Splitchain plays the role of a consensus chain. By construction the consensus committee of C_i contains V/N validators. (Note that Section **??** presents a partial attribution policy that can replace the one presented in Section **??**, when SplitChain is made of sufficiently many chains). For performance reasons, V/N validators cannot all be involved in each consensus execution. The solution we propose relies on a particular asynchronous consensus called the merge consensus algorithm [**?**]. This algorithm leverages the cryptographic sortition lottery introduced by Algorand [**?**] to elect, in a non-interactive and private way, a subset of the committee members. This subset has a bounded expected size that is large enough to handle adversarial behaviors, but independent of the size of the consensus committee. The consensus algorithm runs a series of asynchronous rounds, such that at each round, a new subset of validators is elected via cryptographic sortition. The algorithm ensures, with high probability (whp), that all the transactions proposed by honest validators at the beginning of a consensus execution are included in a block after a finite number of rounds.

4.3 Leveraging a large number of chains

As explained in Section **??**, all reference chains send their latest chaining block containing the Merkle root of their reference set to all the consensus chains. This allows the consensus committee of each chain to be updated with V/N^2 attributed validators. Once these sets have been received, consensus executions can be triggered (see Section **??**). We call this policy the *total attribution policy*. It ensures that consensus committees cannot be compromised by a (μ, δ)-adversary with overwhelming probability (see Theorem **??**). However, it introduces significant delays in the presence of a large number of chains.

When the number of chains is large enough (i.e., $N > 10$) the following*partial attribution* policy is more appropriate. A random subset of S reference chains is selected among the N chains of SplitChain. Each consensus committee contains $S(V/N^2)$ validators instead of $N(V/N^2)$. The S new reference sets of the next consensus committee are selected as follows: Let L be the list of chains included in the latest fingerprint of C_i. List L is randomly shuffled using the random seed of the latest chaining block of C_i. The first S chains in L become the reference chains providing the new reference sets of the next consensus of C_i.

For safety reasons, partial attribution cannot be used in the presence of only a few reference chains, as the (μ, δ)-adversary can concentrate its power to manipulate these few chains. Theorem **??** provides a lower bound S_{min} on S as a function of the total number of reference chains. Due to asynchrony and because the consensus committee of a chain C_i only waits for S reference sets to initiate the consensus, the chaining block of a chain C_j listed inside the fingerprint of the latest chaining block of C_i may be older than the actual latest chaining block of C_j. We call $\rho(N, S)$ the difference between the latest chaining block index of C_j and the latest chaining block index of C_j mentioned inside the fingerprint of C_i. To prevent the adversary from taking advantage of $\rho(N, S)$ to corrupt the new reference sets of C_i, we require its adaptivity to be reduced to $\delta+\rho(N, S)$ and the duration of a validator's membership to be reduced from $\delta - 1$ to $\delta - \rho(N, S)$. So, for the partial attribution policy, we consider a $(\mu, \delta + \rho(N, S))$-adaptive adversary. To bound the value of $\rho(N, S)$ with high probability, the consensus committee of C_i is required to wait for the delivery of any new block b appearing in the fingerprint of the chaining blocks of one of the S new reference committees. Similarly, any new chaining block inside b's fingerprint must be delivered by C_i's committee before initiating a new consensus execution. Theorem **??** provides an upper bound on $\rho(N, S)$.

4.4 Pruning and verifying transactions

It is important to limit the amount of data stored by validators to prevent bootstrapping costs from linearly increasing with the number of blocks in a chain. We propose a pruning method that guarantees near-constant bootstrapping costs. Specifically, each validator v of the consensus committee of $C_i^{\text{cons}(v)}$ stores only all the cue blocks of $C_i^{\text{cons}(v)}$ as checkpoints and only the latest k chaining and transaction blocks of $C_i^{\text{cons}(v)}$. This allows v to respond to users that wish to

verify old transactions that were validated in the latest k blocks. Users wishing to prove the inclusion of old transactions (i.e., those belonging to transaction blocks $b_i, \dots b_\ell$, that have been deleted) must locally store the chaining blocks between $b_i, \dots b_\ell$ and the next cue block. Cryptographic links between consecutive chaining blocks are then enough to recursively prove the existence of a pruned chaining block. The inclusion of the transaction is proven as per usual by the user against the Merkle root of the transactions contained in the chaining block. Each validator also stores the latest cue block of all the other chains: this supports the validator attribution strategy described in Section **??**, as cue blocks contain the initialization proofs of validators.

4.5 Cross-chain transactions

Splitchain's transactions take place between any two user accounts.

Definition 7 (Cross-chain transactions). *When both accounts of a transaction are not managed by the same chain, a transaction is said to be cross-chain.*

The chain in charge of handling a transaction is the one whose label prefixes the emitter account (denoted by C_{send} in the following). Cross-chain transactions are handled in two steps: the withdrawal operation and then the deposit operation. The withdrawal operation occurs when the transaction is inserted in a transaction block of C_{send}. During the creation of the new transaction and chaining blocks, the consensus committee members of C_{send} determine the list of cross-chain transactions inside the transaction block and generate the corresponding *relay transactions*, which they organize as a Merkle tree T. A relay transaction contains the user's initial transaction and the Merkle path of the relay transaction inside T. The Merkle root "relay TX" (see Figure **??**) of T is included into the new chaining block under construction. Consensus committee members of C_{send} then send the relay transaction to the receiver's account chain. As chaining blocks are propagated to all chains, the consensus committee members of the receiver's chain can verify the completion of the withdrawal operation and include the relay transaction in the next transaction block of their chain, which confirms the deposit operation, completing the cross-chain transaction. Note that transactions are always redirected to the chain whose label prefixes the identifier of the emitter account, whereas relay transactions are redirected to the chain of the receiver account.

4.6 Grouping user accounts

Users exchanging on a frequent basis may want to avoid the latency introduced by cross-chain transactions. Thus, we introduce the notion of group accounts that allow users of the same group to always be part of the same chain. A group account is a collection of accounts sharing the same group identifier, i.e. a key hash. When a user u wishes to open an account inside a group gid, u only needs to submit a transaction to a non-existent account (gid, pk_u). Similarly, when a user wishes to withdraw from a group, they can simply submit a transaction to transfer all their account funds elsewhere.

5 Routing

5.1 Hypercube, merging and splitting operations

SplitChain initially starts with a single chain C whose label is the empty chain of bits ε. If during N_{cue} consecutive blocks, the average number of transactions per block exceeds some threshold T_{split}, then C triggers a split. Splitting is an operation that allows a chain of label l to be replaced by two chains of labels $l|0$ and $l|1$, called siblings, each taking over half of the accounts of the original chain. If the original chain C is labeled ε, then both new chains will be labeled 0 and 1 respectively. Conversely, if during N_{cue} consecutive blocks the average number of transactions per block is lower than a threshold T_{merge}, the chain initiates a merge with its sibling chain. Both chains merge in a single one whose label corresponds to the maximal prefix of their previous labels. By design, each chain label is unique. Our topology is inspired by hypercubic networks [**?**]. A hypercube of dimension d contains 2^d vertices. Each vertex is assigned a d-bit label. Two vertices are neighbors if the numerical value of the xor of their labels is equal to a power of 2, i.e. if their labels differ by only one bit. For example, in Figure **??**, vertex 000 is a neighbor of vertices 001, 010 and 100. The number of hops between two vertices can be determined using the Hamming weight of the xor of their labels. The diameter of the hypercube, i.e. the maximum number of hops between the two most distant vertices, is d. We use the following approach to match each chain to one or more vertices of a hypercube: Let k be the number of bits of the longest chain label. Splitchain conforms to a subgraph of a hypercube of dimension k, where each chain whose label has exactly k bits is mapped to the vertex of the same label, and each chain whose label contains strictly less than k bits is mapped to all vertices whose labels are prefixed by the chain's label. Thus, in Figure **??**, chain 01 is mapped to vertices 010 and 011. As a result, when all chain labels are k bits long, the network corresponds to a hypercube of dimension k. A hypercube of dimension k can be constructed recursively by connecting two hypercubes of dimension $k-1$. This allows the dimension of the hypercube to be changed dynamically with k: if k increases, each vertex of label l of size $k-1$ is divided into two vertices of labels $l|0$ and $l|1$. Figure **??** shows the transition from a hypercube of dimension 2 to a hypercube of dimension 3 after chain 00 splits into (i.e., is replaced by) two chains 000 and 001. Similarly, if k decreases, vertices are merged in the same way as for chains. This ensures that no vertex represents two chains simultaneously.

5.2 Core validators

To limit the number of messages transmitted by validators and hence to improve the usage of network resources, we introduce the notion of core validators. The number of core validators is chosen to ensure that at least one honest validator belongs to the core. Election of core validators is as follows. When validator v is attributed to the consensus committee of $C^{\text{cons}(v)}$, v triggers twice the cryptographic sortition lotery at round 0. Once for determining whether it will executes round 0 of the merge-consensus algorithm (see Section **??**) and a second

time to determine whether it will be part of $C^{\text{cons}(v)}$'s core. Both invocations of the cryptographic sortition lotery use the same parameters except for parameter τ, which represents the expected number of successful winners, so that core validators represent a subset of the validators elected for round 0 of the merge consensus. If successful, v is part of the core of $C^{\text{cons(v)}}$ until its consensus credentials expire, which corresponds to the duration that represents $\delta - 1$ consecutive consensus executions (v tries to be elected in the core of $C^{\text{cons(v)}}$ only once during the lifespan of its credentials). Once elected, core validators send their proof of election along with their consensus messages, guaranteeing with any high probability that all committee members of $C^{\text{cons}(v)}$ will agree on the list of core validators at the end of the merge consensus execution. The hash of the list of new core validators will be included in the chaining block decided as the outcome of the merge consensus (see Figure **??**). Core validators are in charge of executing the routing protocol. They maintain two routing tables: A core routing table containing the list of the core validators of neighboring chains for cross-chain communication, and a consensus table containing the list of their consensus committee members. By using their core routing table, core validators forward transactions to the core validators of the neighboring chain whose label is closest to the transaction's destination. On the other hand, the consensus table is used to broadcast messages within their consensus committee. Only core validators keep track of the list of committee members.

5.3 Core routing table

Definition 8 (Core routing table). *The core routing table of chain C_i, $1 \leq i \leq N$, contains the list of the core validators of C_i's neighbouring chains.*

Core routing tables, denoted by RT^{core}, are maintained by core validators and contain the lists of core validators of all of its neighboring chains in the hypercube. For any two neighboring chains C_i and C_j, $RT_i^{core}[j]$ contains a linked list made of at most $\delta - 1$ elements. The k-th element of $RT_i^{core}[j]$, $k \leq \delta - 1$, contains the k-th most recent core proof of C_j and points to the core validators it proves (see Figure **??**). Upon receipt of a new chaining block b of C_j, core validators of C_i create and insert in block order a new element containing the core proof of b. Note that the core validators of C_j send separately the set of new core validators and chaining block b. Once received, it is verified using the core proof of b and then linked to the new element.

5.4 Consensus table

Definition 9 (Consensus routing table). *The consensus routing table of a chain C_i contains the list of validators of the consensus committee of C_i.*

Core validators also keep track of the consensus committee members. All intra-chain communication are handled by core validators, reducing the number of messages within a chain. This also isolates intra-chain communications from the

overall network, allowing chains to be added without overloading SplitChain. A consensus routing table, as shown in Figure **??**, contains the list of consensus committee members sorted by reference chains and chronological order of attribution proof (the Merkle root called "Credentials"). The structure is composed of N linked lists of at most $\delta - 1$ elements, the i-th element of the j-th linked list containing the i-th most recent attribution proof of the validators referenced by the chain C_j. Each element points to the list of attributed validators. This list is updated upon receipt of validators "join" requests. When a core validator receives a new chaining block b from some chain C, it inserts in block order a new element containing b's attribution proof and removes the $\delta-1$ element if needed. This allows the gradual replacement of validators with old credentials, to prevent the (μ, δ)-adversary from compromising consensus committees. In addition, organizing validators with the proof of attribution of the chaining blocks ensures that when a proof expires, all the validator credentials proven by the proof also expire, whether these validators ever participated to the merge consensus executions or not. This prevents the adversary from withholding credentials to build up enough consensus credentials to take over a chain.

6 Security Analysis

In this section we analyze the security of SplitChain. Specifically we first show that an adversary cannot tamper or predict randomness of block seeds, and then we evaluate the probability of corruption of a chain's consensus committee as well as the probability of corruption of a chain's routing core. For the purpose of SplitChain, we can assume that at any time the total number of validators is arbitrarily large. For space constraints, all the proofs are presented in Appendix **??**.

6.1 Randomness creation

The security analysis of SplitChain relies on the assumption that for any chain C of SplitChain, and for any instantiation $k \geq 1$ of the merge consensus, the seeds of the chaining blocks preceding the k-th one of C have been generated in an unbiased and unpredictable way. Theorem **??** is essential to prevent the adaptive adversary from manipulating the outcome of the cryptographic sortition lotery.

Theorem 1. *Let $\epsilon \in (0, 1)$ be the security parameter of SplitChain. The seed of any chaining block cannot be tampered with or predicted in advance by the adversary with any high probability $1 - \epsilon$.*

6.2 Variation in size of reference sets

Lemma **??** shows how fairly the attribution algorithm of SplitChain (i.e., Algorithm **??**) distributes validators between the different consensus committees.

Lemma 1. *Let $E_1, E_2, ..., E_N$ be the reference sets produced by Algorithm **??**. Then $\forall j, k \in \{1, ..., N\}$, $| \, |E_j| - |E_k| \, | \leq 1$.*

6.3 Safety of consensus committees

To ensure that the (μ, δ)-adversary cannot take over the consensus execution of any consensus chain C, the consensus committee of C must contain less than a third of corrupted validators. We examine the probability of corruption of the consensus committee of an arbitrary chain C for both the total attribution policy and the partial one. Recall that μ represents the proportion of malicious validators in SplitChain and N represents the current number of chains of SplitChain.

Total attribution As long as the proportion μ of corrupted validators in SplitChain is less than 1/3, Lemma **??** shows that the (μ, δ)-adaptive adversary must distribute its corrupted validators evenly in the reference sets.

Lemma 2. *The probability of corruption of a consensus committee by the (μ, δ)-adaptive adversary is maximized when corrupted validators are equally distributed among all reference chains.*

Theorem **??** provides the probability p_c of corruption of a consensus committee when the total attribution policy is applied.

Theorem 2. *For any security parameter $\epsilon \in (0, 1)$, for any proportion of corrupted validators $\mu < 1/3$, there exists a finite consensus committee size c_{min} such that $\forall c \geq c_{min}$, the probability p_c that a committee of c validators is corrupted satisfies $p_c < \epsilon$, where $p_c = 1 - \sum_{\ell=0}^{\lfloor c/3 \rfloor} \binom{c}{\ell} \mu^{\ell}(1-\mu)^{c-\ell}$.*

Figure **??** illustrates Theorem **??** for different values of μ and ϵ.

Partial attribution We prove that the partial attribution is secure in presence of a $(\mu, \delta + \rho)$-adversary. Let S represent the number of randomly selected reference chains among the N chains of SplitChain. Theorem **??** gives the probability of corruption p_S of a partially attributed consensus committee. It provides a lower bound S_{min} on the number of reference chains S such that a consensus committee built with the partial attribution policy using $S \geq S_{min}$ reference chains has probability less than ϵ to be corrupted.

Theorem 3. *For any safety parameter $\epsilon \in (0, 1)$, for any proportion of corrupted validators $\mu < 1/3$, there exists a finite number of reference sets S_{min} such that $\forall S \geq S_{min}$, the probability p_S that a consensus committee composed of S reference sets is corrupted satisfies $p_S < \epsilon$, with $p_S = 1 - \sum_{k=0}^{\lfloor S/3 \rfloor)} \binom{S}{k} \mu^{k}(1-\mu)^{S-k}$.*

Figure **??** displays the value of S_{min}/N for $\epsilon = 10^{-6}$ as a function of the adversary proportion μ in SplitChain and for different values of N. The value of S_{min} decreases when the number N of chains of the system increases. However, when the adversary proportion μ becomes very close to 1/3, S_{min}/N tends to 1 and total attribution is needed. Our experiments run on Ethereum show that both policies can be safely applied on Ethereum (see Appendix **??**).

6.4 Resistance against an adaptive adversary

The objective of this section is to analyze whether the adversary can benefit from the partial attribution policy to corrupt a targeted consensus committee via the manipulation of a given reference set S_{j*} of a chain C_{j*}. Specifically, let $\mathcal{C}(t)$ be the consensus committee of chain C at time t, and $\mathcal{B}(t) = \{S_1, \ldots, S_{j*}, \ldots, S_N\}$ be the set of the last reference sets sent by the N chains of SplitChain that have been received by $\mathcal{C}(t)$. From the partial attribution policy (see Section **??**), $\mathcal{C}(t)$ randomly selects S sets from $\mathcal{B}(t)$ to determine the composition of the next consensus committee $\mathcal{C}(t+1)$ at time $t+1$. Consensus committee $\mathcal{C}(t+1)$ on its turn will randomly select S sets from $\mathcal{B}(t+1) = \{S_1^{(1)}, \ldots, S_{j*}, \ldots, S_N^{(1)}\}$, where $S_i^{(1)}$ represents the last reference set received by $\mathcal{C}(t+1)$ and sent by C_i. Note that, by the asynchrony of the system, all communications from the chains may be arbitrarily long, delaying accordingly the receipt of reference sets, and in particular the new reference set of C_{j*}. Thus $\mathcal{C}(t+1)$ may still consider the obsolete reference set S_{j*} as the latest reference set of C_{j*}. Let $\rho_{(N,S)}$ be the upper bound on the difference between the chaining block index of the obsolete reference set S_{j*} and the latest chaining block index of C_{j*}. Consensus committee $\mathcal{C}(t+\rho_{(N,S)})$ will randomly select S reference sets from $\mathcal{B}(t+\rho_{(N,S)}) = \{S_1^{(\rho_{(N,S)})}, \ldots, S_{j*}, \ldots, S_N^{(\rho_{(N,S)})}\}$. Suppose that reference set S_{j*} is never selected during the first $\rho_{(N,S)} - 1$ random selections. Recall that for the partial attribution policy (see Section **??**), we assume a $(\mu, \delta + \rho_{(N,S)})$-adaptive adversary. Were reference set S_{j*} selected after $\rho_{(N,S)}$ validator attributions, the adversary could already have corrupted the validators of S_{j*}. Let $\chi_{\rho_{(N,S)}}$ be the probability that a consensus committee uses a reference set that has been obsolete for at least $\rho_{(N,S)}$ consecutive validator attributions. Theorem **??** shows that $\chi_{\rho_{(N,S)}}$ is less than the security parameter ϵ. Figure **??** illustrates that $\rho_{(100,24)} = 5$.

Theorem 4. *$\forall N, \forall S \leq N$, $\exists \rho_{(N,S)} > 0$ such that $\chi_{\rho_{(N,S)}} < \epsilon$.*

6.5 Safety of the election of core validators

Validators of the core of a chain are crucial for intra- and cross-chain communications. There must always be at least one honest core validator to ensure the routing of messages in a chain. In the following, T represents the lifespan of consensus credentials, $\mu' = \mu(T)/(T-1)$ refers to the maximum proportion of corrupted validators eligible for the core and c is the number of validators of any consensus committee. We use $1 \leq r \leq T$ to designate the T last consensus of a chain and refer to the number of validators that joined the consensus committee during consensus r as c_r. Theorem **??** shows that there exists a finite core size above which the probability p_{core} that the core is corrupted is smaller than ϵ.

Theorem 5. *For any safety parameter $\epsilon \in (0,1)$, for an adaptive adversary with $\delta \geq 3$, there exists a finite core size τ_{min} such that $\forall \tau \geq \tau_{min}$, the probability p_{core} that a core of τ validators is corrupted satisfies $p_{core} < \epsilon$, with $p_{core} = (1 - \tau/c)^{(1-\mu')(c-c_T)}$.*

Figure **??** plots τ_{min} as a function of the consensus committee size c for different values of δ. As observed, τ_{min} quickly stabilizes as c increases. We also notice that τ_{min} decreases very rapidly when δ increases.

7 Related Work

While several sharded blockchains have been proposed in recent years, we focus on previous works that have introduced new concepts that have influenced the design of SplitChain.

Elastico [**?**] is the first sharded blockchain. Its validators are divided into shards, each one creating a block of transactions, which are then aggregated into a "global-block" to add to the system's unique blockchain. The shards are renewed after each global-block, ensuring strong safety against adaptive adversaries at the expense of synchronization and storage costs, i.e., validators store the entire system. As a safeguard against Sybil attacks, each validator must solve a PoW puzzle, whose solution also determines which shard it belongs to. Elastico cannot ensure atomicity in cross-shard transactions [**?**], leading to permanent fund locking. Its attribution protocol coupled with its small shard size (approx. 100 validators) yields a very high corruption probability of 2.76% [**?**] per shard per block. In comparison, SplitChain splits computation and storage between the different chains at the cost of a slower adaptive adversary. SplitChain implements a pruning mechanism for the blocks of the chains and gradually renews the validators of the chains, limiting the bootstrapping overhead of new validators. SplitChain supports cross-chain transactions natively and includes a routing protocol designed for chains to route transactions in a logarithmic number of hops in proportion to the total number of chains in the system. Finally, we do not rely on PoW, which requires a synchronous communication model and whose use is controversial because of its excessive energy consumption.

Omniledger [**?**] improves upon Elastico. It uses a more scalable consensus algorithm, increasing the size of its shards, thus reducing the probability of their corruption. It features shards with separate ledgers to better distribute storage costs and adapts classic distributed checkpointing principles to prune shard ledgers. It provides a synchronous lock/unlock client-driven mechanism to handle cross-shard transactions, although at the expense of lightweight-client compatibility and significant latency and safety issues [**?**]. The UTXO model is not adapted to sharding, as its cross-chain transactions can consume UTXOs stored in different chains, having a significant impact on throughput as the number of shards increases. Omniledger reduces the cost of shard reconfiguration by bounding the number of validators shuffled in each of its day-long epochs, however, Omniledger requires a global blockchain for validator allocation. In contrast, SplitChain shuffles a small number of validators of a chain after each block to withstand an adversary adaptive over a small, configurable number of consensus instances and does not require any global blockchain to manage validator identities. Validators can handle the management and routing of cross-chain transactions autonomously, preserving lightweight-client compatibility.

Zilliqa [**?**] is an account-based sharded blockchain that handles smart contracts. It inherits all the problems of Elastico except for its ability to handle validators on a separate chain. It does not shard transactions storage. Moreover, it cannot provide atomicity for cross-shard transactions.

RapidChain [**?**] is a UTXO-based sharded blockchain that distinguishes itself from Omniledger by featuring a transaction routing protocol inspired by Kademlia, enabling message routing in $\log n$ steps without any special client interaction. To reduce the size of its shards, RapidChain uses a synchronous consensus algorithm tolerating a proportion of $1/2$ Byzantine nodes. RapidChain thus inherits problems of Elastico and Omniledger, namely the use of PoW for validator enrollment (Elastico) and the use of the UTXO model (Omniledger). On the other hand, Splitchain is asynchronous, does not require PoW and routes transactions through a small subset of validators (the core) thereby significantly reducing message load.

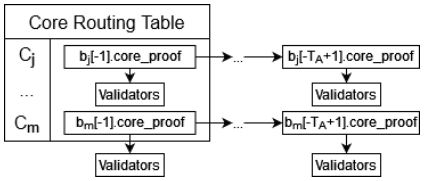
Monoxide [**?**] is the first sharded blockchain to implement the use of PoW for shard consensus. It allows each miner to finalize and propose new transaction blocks to multiple shards simultaneously, amplifying and distributing their mining capabilities within the system. Its cross-shard transactions are handled in a lock-free manner. However, to guarantee the safety of Monoxide, the majority of miners are required to work for most if not all shards. This causes centralization problems and contradicts the load-distribution principle of sharding, requiring large throughput, storage and computation costs from miners. This behavior causes Monoxide to resemble more to a parallelization solution than to an actual sharded system like SplitChain.

8 Conclusion

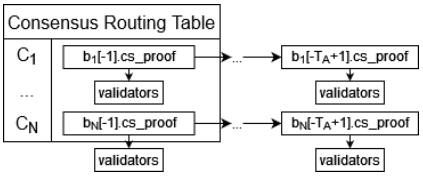
We have presented SplitChain, a protocol supporting the creation of scalable account-based blockchains without undermining decentralization and security. SplitChain distinguishes itself from other sharded blockchains by minimizing the synchronization constraints among shards while maintaining security guarantees. Specifically, SplitChain is the first permissionless sharded blockchain that does not require a dedicated shard or a global blockchain to attribute validators to their consensus chain. This avoids the need for a global reconfiguration of the shards each time a new batch of validators is added to the system. A dedicated routing protocol enables transactions to be redirected between shards with a low number of hops and messages. Finally, SplitChain dynamically adapts the number of shards to the system load to avoid over-dimensioning issues encountered in static sharding-based solutions. Further research will investigate the practical performance of SplitChain through its implementation and the potential use of sharding to enhance user privacy.

A Figures

Fig. 1: Composition of a chaining block (bottom fields only belong to cue blocks.)



(a) Structure of the core routing table



(b) Structure of the consensus routing table

Fig. 2: Routing tables

Fig. 3: Mapping of vertices and chain labels after a chain split causing an increase in hypercube dimension.

B Algorithms

Algorithm 1: Initializing a validator account

input : The validator account registering transaction tx.

function `initialize`(tx, pk_{val})
- send(tx).to_any()
- $b \leftarrow$ `wait_for_validation(`tx`)`
- $C_{\text{init}} \leftarrow b.chain$
- $b_{\text{init}} \leftarrow$ `wait_for_cue_block(`C_{init}`)`
- $tx_{\text{init}} \leftarrow (\text{"}init\text{"}, pk_{val})$
- $ack, \pi_{\text{init}} \leftarrow$ `wait_for_ack(`tx_{init}`)`
- $core\_table[C_{\text{init}}] \leftarrow ack.table$
- **return** $\pi_{\text{init}}, b_{\text{init}}, C_{\text{init}}, core\_table$

output: The validator's initialization proof and cue block and its initialization chain's label and core table.

Algorithm 2: Referencing a validator

```
function reference(π_init, core_init)
    pk_cons, sk_cons ← generate_keys()
    tx ← ("credential", π_init, pk_cons)
    send(tx).to(core_init)
    b, π_ref ← wait_for_validation(tx)
    C_ref ← b.chain
    return π_ref, pk_cons, sk_cons, C_ref
```

output: The validator's referencing proof.

Algorithm 3: Attribution algorithm

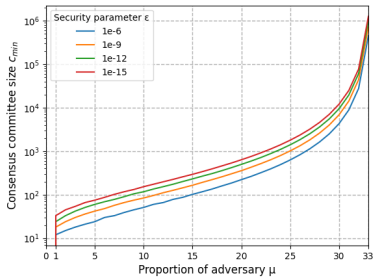
input : The list of validators L to attribute to chains, the number N of chains, a random seed *seed*.

```
function attribution(L, N, seed)
    L ← shuffle(L, seed)
    sets ← []
    α ← ⌊|L|/N⌋
    r ← |L| − N × α
    for i ← 0 to N − 1 do
        sets.append([])
        for j ← 0 to α − 1 do
            sets[i].append(L.pop())
        if r > 0 then
            sets[i].append(L.pop())
            r ← r − 1
    return sets
```

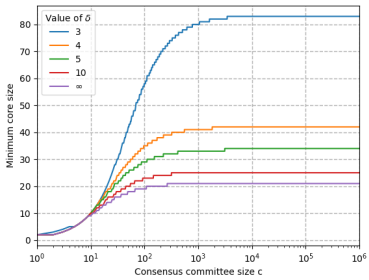
Algorithm 4: Join a chain for consensus

input : The reference proof π_{ref}.

```
function join_consensus(π_ref, core_table)
    tx ← ("join", π_ref)
    send(tx).to_core(C_ref)
    ack ← wait_for_ack(tx)
    core_table[C_cons] ← ack.table
    The validator bootstraps itself to the chain using the list of nodes
      provided in the ack message
    return core_table
```

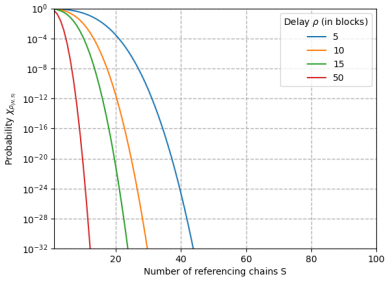


(a) Minimum number of validators c_{min} of a consensus committee as a function of μ for different safety probabilities $1 - \epsilon$.

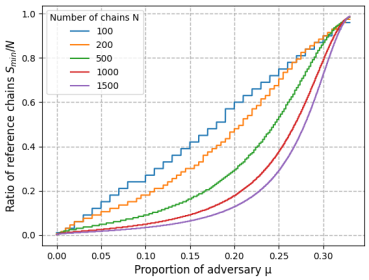


(b) Minimum core size τ_{min} as a function of the consensus committee size c for different values of adversary adaptivity δ

Fig. 4



(a) Probability $\chi_{\rho_{(N,S)}}$ as a function of the number of reference chains S and the delay in blocks ρ for $N = 100$ chains.



(b) Required proportion of reference chains to achieve partial attribution with probability $1 - 10^{-6}$ as a function of μ for different values of N.

Fig. 5

C Proofs

C.1 Randomness creation

Theorem ?? Let $\epsilon \in (0, 1)$ be the security parameter of SplitChain. The seed of any chaining block b cannot be tampered with or predicted in advance by the adversary with any high probability $1 - \epsilon$.

Proof. We assume the existence of a preceding chaining block $C[k-1]$ with a proper pseudorandom seed s_0. This block may be the genesis block of SplitChain if k is the first consensus of the system. Similarly, if the system consists of multiple chains, we suppose that the seeds of the preceding blocks of these chains are also pseudorandom and unpredictible.

The attribution of the new validators of the consensus committee of $C[k]$ is based on the last seeds of its reference chains. As these seeds are both pseudorandom and unpredictable, the result of the attribution of the new validators of $C[k]$ using these seeds is also pseudorandom and unpredictable. As the adversary needs δ blocks of delay to corrupt validators, it is not possible for it to target the validators of the reference committees quickly enough to influence the randomness of the attribution. Furthermore, in the case of partial attribution, the adversary cannot predict which of the system's chains will serve as the reference chains of the consensus committee of $C[k]$, as they are also randomly chosen using the seed of $C[k-1]$. Thus, if the preceding seeds are unpredictable, the adversary cannot temper the randomness of the attribution of new validators for $C[k]$'s consensus committee.

Furthermore, the verifiable random function (VRF) used in the pseudorandom sortition algorithm (see Section **??**) run by new validators of the consensus committee to determine if they are part of the core uses the seed of $C[k-1]$ and the private key of the validators. As the seed used by the sortition function is both pseudorandom and unpredictable, and the core contains at least one honest validator with high probability $1 - \epsilon$, the new seed of $C[k]$ obtained by hashing the concatenation of the core validator's public keys is both pseudorandom and unpredictable. Thus, the adversary cannot predict or manipulate the value of the new seed in advance. □

C.2 Fair attribution of reference sets

Lemma ?? Let $E_1, E_2, ..., E_N$ be the reference sets produced by Algorithm **??**. Then $\forall j, k \in \{1, ..., N\}$, $| \, |E_j| - |E_k| \, | \leq 1$.

Proof. Let N and L be respectively the number of reference sets to be produced and the set of validators to be attributed. The algorithm first assigns $\alpha = \lfloor |L|/N \rfloor$ validators to each set. This makes $r = |L| - \alpha N < N$ validators to be attributed. Then for the r remaining validators, the algorithm assigns each of them to a different set. This results in $0 \leq r < N$ sets of $\alpha + 1$ validators and $N - r$ sets of α validators. □

C.3 Consensus committee safety

Let $A = \mu V$ be the total number of corrupted validators in SplitChain. Let A_i be the number of corrupted validators in the reference set of C_i, $1 \leq i \leq N$. We have $A = \sum_{i=1}^{N} A_i$. Recall that the attribution of any validator lasts for $\delta - 1$ consecutive consensus. Let A_i^k be the random variable that represents the maximal number of malicious validators attributed by the $(\ell + k)$-th instance of Algorithm **??** executed in C_i, for any $\ell \geq 1$ and $k \in [1, \delta - 1]$. We have $A_i = \sum_{k=1}^{\delta-1} A_i^k$.

Each reference chain C_i, $1 \leq i \leq N$, sends a new reference set of validators to a chain C_j to update the consensus committee membership of C_j. Let $X_{i,j}$ be the random variable that represents the number of corrupted validators assigned by C_i to the consensus committee of C_j. We have $0 \leq X_{i,j} \leq A_i$. Specifically, the number of new corrupted validators $X_{i,j}^k$ attributed by C_i to the $(\ell' + k)$-th consensus committee of C_j, for any $\ell' \geq 1$ and $k \in [1, \delta - 1]$, is less than or equal to A_i^k and we have $X_{i,j} = \sum_{k=1}^{\delta-1} X_{i,j}^k$.

Let Y_j be the random variable that represents the number of corrupted validators in the consensus committee of a chain C_j. We have that Y_j is the sum of the corrupted validators of all the reference sets attributed to C_j by its reference chains.

Lemma C.1 *$X_{i,j}^k$ follows a hypergeometric distribution with parameters $V/(N(\delta-1))$, $V/((\delta-1)N^2)$ and A_i^k.*

Proof. The reference set attributed by the $(\ell + k)$-th instance of Algorithm **??** in C_i to the consensus committee of C_j, for any $\ell \geq 1$ and $k \in [1, \delta - 1]$, is a dichotomous population of size $V/(N(\delta-1))$ composed of A_i^k corrupted validators and $(V/(N(\delta-1))) - A_i^k$ honest validators. Each validator in this population can only be allocated to one consensus committee, hence the sampling is drawn without replacement. Finally, validators are evenly allocated between the N consensus chains, so each attribution subset contains $V/((\delta-1)N^2)$ validators. Hence, $X_{i,j}^k \sim H(V/(N(\delta-1)), V/((\delta-1)N^2), A_i^k)$. □

C.4 Consensus committee safety with the total attribution policy

Recall that in the total attribution policy, the consensus committee of any chain C_j of SplitChain is fed with the reference sets of each chain of SplitChain. Thus we have $Y_j = \sum_{i=1}^{N} X_{i,j} = \sum_{i=1}^{N} \sum_{k=1}^{(\delta-1)} X_{i,j}^k$.

Lemma ?? The probability of corruption of a consensus committee by the (μ, δ)-adaptive adversary is maximized when corrupted validators are equally distributed among all reference chains, i.e., $\forall i$, $\forall k$, $A_i^k = A/(N(\delta-1))$.

Proof. By definition of Y_j and by applying lemma **??**, we have

$$Y_j = \sum_{i=1}^{N} X_{i,j} = \sum_{i=1}^{N} \sum_{k=1}^{(\delta-1)} H\left(\frac{V}{N(\delta-1)}, \frac{V}{(\delta-1)N^2}, A_i^k\right), \tag{1}$$

and the expectation of Y_j is given by

$$E(Y_j) = \sum_{i=1}^{N} \sum_{k=1}^{(\delta-1)} \left(\frac{V}{(\delta-1)N^2}\right)\left(\frac{(\delta-1)NA_i^k}{V}\right) = \sum_{i=1}^{N} \sum_{k=1}^{(\delta-1)} \frac{A_i^k}{N} = \frac{A}{N}. \quad (2)$$

The average proportion of the (μ, δ)-adaptive adversary in a consensus committee is therefore $(A/N)/(V/N) = A/V = \mu$. Note that this value is independent from A_i^k. Therefore, the adversary can only maximize its chances of corrupting the consensus committee of C_j by increasing the variance of Y_j. The variables $X_{i,j}^k$ are drawn from disjoint populations, and are therefore independent. Thus, the variance of Y_j is given by summing the variances of random variables $X_{i,j}^k$:

$$\begin{aligned}
\mathrm{Var}(Y_j) &= \sum_{i=1}^{N} \sum_{k=1}^{(\delta-1)} \frac{V}{(\delta-1)N^2} \frac{(\delta-1)NA_i^k}{V} \left(1 - \frac{(\delta-1)NA_i^k}{V}\right) \frac{\frac{V}{N(\delta-1)} - \frac{V}{(\delta-1)N^2}}{\frac{V}{N(\delta-1)} - 1} \\
&= \sum_{i=1}^{N} \sum_{k=1}^{(\delta-1)} \frac{A_i^k}{N} \left(1 - \frac{(\delta-1)NA_i^k}{V}\right) \frac{\frac{V(N-1)}{(\delta-1)N^2}}{\frac{V}{N(\delta-1)} - 1} \\
&= \sum_{i=1}^{N} \sum_{k=1}^{(\delta-1)} \left(\frac{A_i^k}{N} - \frac{(\delta-1){A_i^k}^2}{V}\right) \frac{V\,(N-1)}{N(\delta-1)\,(V-N)} \\
&= \frac{V\,(N-1)}{N(\delta-1)\,(V-N)} \left(\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{(\delta-1)} A_i^k - \frac{(\delta-1)}{V} \sum_{i=1}^{N} \sum_{k=1}^{(\delta-1)} {A_i^k}^2\right) \\
&= \frac{V\,(N-1)}{N(\delta-1)\,(V-N)} \left(\frac{A}{N} - \frac{(\delta-1)}{V} \sum_{i=1}^{N} \sum_{k=1}^{(\delta-1)} {A_i^k}^2\right). \quad (3)
\end{aligned}$$

To maximize $\mathrm{Var}(Y_j)$, the (μ, δ)-adversary must choose the A_i^k so that to minimize $\sum_{i=1}^{N} {A_i^k}^2$, with $0 \leq A_i^k \leq V/(N(\delta-1))$ and $\sum_{i=1}^{N} \sum_{k=1}^{(\delta-1)} A_i^k = A$. Intuitively, $\sum_{i=1}^{N} \sum_{k=1}^{(\delta-1)} {A_i^k}^2$ is minimal when $A_i^k = A/(N(\delta-1))$. We formulate this hypothesis by the following inequality, where v_i^k represents the deviation of A_i^k from $A/(N(\delta-1))$, such that $-A/(N(\delta-1)) \leq v_i^k \leq V/(N(\delta-1)) - A/(N(\delta-$

1)) and $\sum_{i=1}^{N}\sum_{k=1}^{(\delta-1)} v_i^k = 0$. We have:

$$\sum_{i=1}^{N}\sum_{k=1}^{(\delta-1)} {A_i^k}^2 \leq \sum_{i=1}^{N}\sum_{k=1}^{(\delta-1)} (\frac{A}{N(\delta-1)})^2$$

$$< \sum_{i=1}^{N} (\frac{A}{N(\delta-1)} + v_i^1)^2 + (\frac{A}{N(\delta-1)} + v_i^2)^2 + ... + (\frac{A}{N(\delta-1)} + v_i^{(\delta-1)})^2$$

$$N(\delta-1)(\frac{A}{N(\delta-1)})^2 < \sum_{i=1}^{N}(\delta-1)(\frac{A}{N(\delta-1)})^2 + \sum_{k=1}^{(\delta-1)} {v_i^k}^2 + 2\frac{A}{N(\delta-1)}\sum_{k=1}^{(\delta-1)} v_i^k$$

$$\frac{A^2}{N(\delta-1)} < \frac{A^2}{N(\delta-1)} + \sum_{i=1}^{N}\sum_{k=1}^{(\delta-1)} {v_i^k}^2$$

$$0 < \sum_{i=1}^{N}\sum_{k=1}^{(\delta-1)} {v_i^k}^2 \tag{4}$$

We notice that inequality **??** stands when any deviation v_i^k is different from 0, that is, any other distribution than $A_i^k = A/(N(\delta-1))$ results in a greater value of the sum and reduces the variance of Y_j. Therefore, to maximize the variance of Y_j, i.e., to maximize the corruption probability of the consensus committee of C_j, malicious validators must be evenly distributed across the k executions of Algorithm **??**. □

Lemma C.2 *The probability p_c that a consensus committee of chain C_j is corrupted by the adversary is given by*

$$p_c = P\left(Y_j > \lfloor\frac{c}{3}\rfloor\right) = 1 - \sum_{l=0}^{\lfloor\frac{c}{3}\rfloor}\binom{c}{l}(\mu)^l(1-\mu)^{c-l} \tag{5}$$

Proof. A consensus committee is corrupted if the sum of the adversary's validators is greater than one third of the chain. By applying lemma **??** to equation **??**, we can express the probability of corruption p_c of a consensus committee as:

$$\begin{aligned} p_c &= P\left(Y_j > \lfloor\frac{V}{3N}\rfloor\right) \\ &= 1 - P\left(\sum_{i=1}^{N}\sum_{k=1}^{(\delta-1)} H\left(\frac{V}{N(\delta-1)}, \frac{V}{(\delta-1)N^2}, \frac{A}{N(\delta-1)}\right)\right) \\ &\leq \left\lfloor\frac{V}{3N}\right\rfloor \end{aligned} \tag{6}$$

The binomial distribution is a good approximation of the hypergeometric distribution when the population is sufficiently larger than the sample size, which

is commonly accepted as a sampling fraction of 0.1 [**?**]. That is, $H(m,n,p) \approx B(n,p/m)$ when $n/m < 0.1$. In our case, $(V/((\delta-1)N^2))/(V/(N(\delta-1))) < 0.1$ imposes that $N > 10$. Thus, we assume that the number of chains N is always greater than 10. By applying this approximation to equation **??**, we can simplify our equation:

$$\begin{aligned} P(Y_j \leq \lfloor \frac{V}{3N} \rfloor) &= \lim_{V\to\infty} P(\sum_{i=1}^{N}\sum_{k=1}^{(\delta-1)} H(\frac{V}{N(\delta-1)}, \frac{V}{(\delta-1)N^2}, \frac{A}{N(\delta-1)}) \leq \lfloor \frac{V}{3N} \rfloor) \\ &= P(\sum_{i=1}^{N}\sum_{k=1}^{(\delta-1)} B(\frac{V}{(\delta-1)N^2}, \frac{A}{V}) \leq \lfloor \frac{V}{3N} \rfloor) \\ &= P(B(\sum_{i=1}^{N}\sum_{k=1}^{(\delta-1)} \frac{V}{(\delta-1)N^2}, \mu) \leq \lfloor \frac{V}{3N} \rfloor) \\ &= P(B(\frac{V}{N}, \mu) \leq \lfloor \frac{V}{3N} \rfloor) \\ &= \sum_{k=0}^{\lfloor \frac{V}{3N} \rfloor} \binom{\frac{V}{N}}{k} (\mu)^k (1-\mu)^{\frac{V}{N}-k} \end{aligned} \tag{7}$$

Let c be the constant representing the minimal number of validators needed to ensure the proper and safe execution of a chain. By replacing V/N with c in equation **??**, we can calculate the probability of corruption of a chain by the adversary:

$$P(Y_j > \lfloor \frac{c}{3} \rfloor) = 1 - \sum_{k=0}^{\lfloor \frac{c}{3} \rfloor} \binom{c}{k} (\mu)^k (1-\mu)^{c-k}$$

□

Theorem ?? For any security parameter $\epsilon \in (0,1)$, for any proportion of corrupted validators $\mu < 1/3$, there exists a finite consensus committee size c_{min} such that $\forall c \geq c_{min}$, the probability p_c that a committee of c validators is corrupted satisfies $p_c < \epsilon$, where $p_c = 1 - \sum_{\ell=0}^{\lfloor c/3 \rfloor} \binom{c}{\ell} \mu^\ell (1-\mu)^{c-\ell}$.

Proof. Probability p_c is given by lemma **??**. Let $c = V/N$ be the number of validators inside a consensus committee, and $\mu = A/V$ the proportion of corrupted validators in SplitChain, with $\mu < 1/3$. We know from Equation **??** that the average number $E(Y_j)$ of corrupted validators in a consensus committee is $A/N = \mu c$. We also know from lemma **??** that $p_c = 1 - P(Y_j \leq \lfloor c/3 \rfloor)$, with $Y_j \sim BinomialDistribution(c, \mu)$.

Let $\mu = 1/3$. Then $E(Y_j) = c/3$. By definition of the statistical mean, we have $\lim_{c\to\infty} P(Y_j \leq E(Y_j)) = 0.5$. Thus, when $\mu = 1/3$, $\lim_{c\to\infty} p_c = 0.5$. However, SplitChain strictly requires that $\mu < 1/3$. As $E(Y_j) < c/3$ when $\mu < 1/3$, $\lim_{c\to\infty} P(Y_j \leq \lfloor c/3 \rfloor) = 1$ and $\lim_{c\to\infty} p_c = 0$. By definition, the cumulative distribution function is monotone increasing, so p_c is monotone decreasing. Thus, there exists a value c_{min}, such that for any $\epsilon < 1$ and $c \geq c_{min}$, $p_c < \epsilon$. □

C.5 Consensus committee safety with the partial attribution

Let S be the number of reference sets required for a chain C_j to update its $(\ell+k)$-th consensus committee before initiating a new consensus execution. Let $R_a^k \geq N$ be the number of referencing sets corrupted by the adversary among the N reference sets proposed to C_j by the chains of SplitChain at some attribution execution $(\ell'+k)$. The number of corrupted reference attributed to the $(\ell+k)$-th consensus committee of C_j is the random variable $S_a^k \leq S$.

Lemma C.3 *The number of corrupted validators Y_j is maximized when the corrupted validators are equally distributed among the R_a reference chains, i.e. $\forall C_i \in R_a, A_i = \mu' V/N$, where μ' is the new proportion of corrupted validator inside the S_a corrupted reference chains, such that $\mu' \geq \mu$.*

Proof. This result is obtained by applying lemma **??** with S referencing chains instead of N and μ' instead of μ. □

Lemma C.4 *S_a follows the hypergeometric distribution with parameters $H(N, R_a, S)$.*

Proof. The reference sets correspond to a dichotomous population of size N composed of R_a corrupted sets and $N - R_a$ honest sets. Each set in this population can only be allocated once to the same consensus committee, hence the sampling is drawn without replacement. Finally, only S sets are chosen at random, thus the number of randomly selected corrupted sets S_a follows the hypergeometric distribution $H(N, R_a, S)$. □

Lemma C.5 *The proportion of malicious validators inside a consensus committee $S_a \times (\mu'/S)$ is maximized when the reference chains targeted by the adversary are fully corrupted, i.e. when $\mu' = 1$.*

Proof. The average value of S_a is: $E(S_a) = S\frac{R_a}{N} = S\frac{(\mu/\mu')N}{N} = S\frac{\mu}{\mu'}$. This corresponds to an average adversary proportion of $E(S_a)\frac{\mu'}{S} = \mu$ corrupted reference validators. Note that this mean value doesn't depend on μ'.

The adversary proportion's variance is:

$$\begin{aligned} V_a(S_a\frac{\mu'}{S}) &= \frac{\mu'^2}{S^2} S \frac{R_a}{N}\frac{N-R_a}{N}\frac{N-S}{N-1} \\ &= \frac{\mu'^2}{S}\frac{\frac{\mu}{\mu'}N}{N}\frac{N-\frac{\mu}{\mu'}N}{N}\frac{N-S}{N-1} \\ &= \frac{\mu'^2}{S}\frac{\mu}{\mu'}(1-\frac{\mu}{\mu'})\frac{N-S}{N-1} \\ &= \frac{\mu}{S}(\mu'-\mu)\frac{N-S}{N-1} \end{aligned}$$

The derivative function of the adversary proportion's variance is $\frac{\partial}{\partial\mu'}V_a(S_a\frac{\mu'}{S}) = \frac{\mu}{S}\frac{N-S}{N-1}$, whose roots are $N = S$ and $\mu = 0$. If $S = N$, then all the available

reference validators are used and the proportion of corrupted reference validators is μ. For $\mu > 0$, $\mu' \leq 1$ and $0 < S < N$, the derivative is always positive. Hence, the adversary proportion's variance is ascending for $\mu \leq \mu' \leq 1$ and reaches its maximum when $\mu' = 1$. □

Lemma C.6 *(Probability of corruption of a partially attributed consensus committee).*

$$p_c = P(Y_j > \lfloor c/3 \rfloor) = P(S_a > \lfloor S/3 \rfloor) = 1 - \sum_{k=0}^{\lfloor S/3 \rfloor)} \binom{S}{k} (\mu)^k (1-\mu)^{n-k} \tag{8}$$

Proof. A consensus committee is corrupted if it contains more than one third of malicious validators. As the committee is composed of S reference sets, this corresponds to $S(V/3N^2)$ validators. By definition of Y_j and by applying lemma **??**, we have:

$$p_c = P(\sum_{i=1}^{S_a} H(\frac{V}{N}, \frac{V}{N^2}, \frac{V}{N}\mu') > \lfloor S\frac{V}{3N^2} \rfloor) \tag{9}$$

Let $\lim_{V,N\to\infty} \frac{V}{N^2} = c$, c being the fixed number of validators within an attribution subset.

$$\begin{aligned}
\lim_{V,N\to\infty} P(Y_j > \lfloor S\frac{V}{3N^2} \rfloor) &= \lim_{V,N\to\infty} P(\sum_{i=1}^{S_a} H(\frac{V}{N}, \frac{V}{N^2}, \frac{V}{N}\mu') > \lfloor S\frac{V}{3N^2} \rfloor) \\
&= \lim_{V,N\to\infty} P(\sum_{i=1}^{S_a} H(\frac{V}{N}, c, \frac{V}{N}\mu') > \lfloor S\frac{c}{3} \rfloor) \\
&= P(\sum_{i=1}^{S_a} B(c, \mu') > \lfloor S\frac{c}{3} \rfloor) \\
&= P(B(S_a c, \mu') > \lfloor S\frac{c}{3} \rfloor)
\end{aligned}$$

We know from lemma **??** that this value is maximized when $\mu' = 1$, thus we obtain $p_c = P(S_a c > \lfloor S(c/3) \rfloor) = P(S_a > \lfloor S/3 \rfloor)$. As a reference set contains at least one validator, i.e. $c \geq 1$, the minimum number of consensus validators of a chain is S. From lemma **??**, we know that $S_a \sim H(N, R_a, S)$. As $\mu' = 1$, $R_a = \mu N$. Thus:

$$\begin{aligned}
p_c &= \lim_{V,N\to\infty} P(S_a > \lfloor S/3 \rfloor) \\
&= \lim_{N\to\infty} P(H(N, \mu N, S) > \lfloor S/3 \rfloor) \\
&= P(B(S, \mu) > \lfloor S/3 \rfloor) \\
&= 1 - \sum_{\lfloor S/3 \rfloor)}^{k=0} \binom{S}{k} (\mu)^k (1-\mu)^{S-k}
\end{aligned}$$

□

Theorem ?? For any security parameter $\epsilon \in (0,1)$, for any proportion of corrupted validators $\mu < 1/3$, there exists a finite number of reference sets S_{min} such that $\forall S \geq S_{min}$, the probability p_S that a consensus committee composed of S reference sets is corrupted satisfies $p_S < \epsilon$, with $p_S = 1 - \sum_{k=0}^{\lfloor S/3 \rfloor)} \binom{S}{k} \mu^k (1-\mu)^{S-k}$.

Proof. We know from lemma **??** that the number of fully corrupted reference sets in a committee is $p_c = P(S_a > \lfloor S/3 \rfloor) = 1 - P(S_a \leq \lfloor S/3 \rfloor)$, with $S_a \sim B(S, \mu)$. The average number of fully corrupted reference sets in a consensus committee is $E(S_a) = S\mu$.

Let $\mu = 1/3$. Then $E(S_a) = S/3$. By definition of the statistical mean, we have $\lim_{S\to\infty} P(S_a \leq E(S_a)) = 0.5$. Thus, when $\mu = 1/3$, $\lim_{S\to\infty} p_c = 0.5$. However, SplitChain strictly requires that $\mu < 1/3$. As $E(S_a) < S/3$ when $\mu < 1/3$, $\lim_{S\to\infty} P(S_a \leq \lfloor S/3 \rfloor) = 1$ and $\lim_{S\to\infty} p_c = 0$. Because the cumulative distribution function is by definition monotone increasing, p_c is monotone decreasing. Thus, there exists a value S_{min} such that for any $\epsilon > 0$ and $S > S_{min}$, $pc < \epsilon$. □

C.6 Partial attribution policy: Probability to corrupt a given consensus set

Lemma C.1. *Let $\chi_{\rho_{(N,S)}}$ be the probability that a consensus committee uses a reference set that has been obsolete for at least $\rho_{(N,S)}$ consecutive validator attributions. Then $\chi_{\rho_{(N,S)}} \leq P(H(N,S,1) = 0)\Big(P(H(N,S,1) = 0)p_{2,S} + P(H(N,S,1) = 1)p_{2,S-1}\Big)$, where H is the hypergeometric distribution and*

$$p_{i,k} = \begin{cases} 1, & \text{if } i > \rho_{(N,S)} \\ 0, & \text{if } k \geq N \\ P(H(N,S,1) = 0) \sum_{j=0}^{min(S,k)} \Big(P(H(N,S,k) = j) & \\ (P(H(N,S,1) = 0))^j p_{i+1,k+S-j}\Big) & \end{cases}$$

Proof. We first assume that all chains are up to date, i.e. at time t, each chain knows the penultimate block $t-1$ of all other chains. Because of the asynchrony, we adopt the worst-case scenario and assume that the index of chain C_1 is only updated by a chain C_2 if C_1 is selected for the partial attribution of C_2, i.e. a new chaining block of C_1 is required to initiate the consensus of C_2.

Let χ_1 be the probability that C does not draw C'. Since all chains are up to date, none of the drawn chains contain block indices unknown to C, therefore $\chi_1 = P(H(N,S,1) = 0$. Given that C can draw itself, the number of new chain block indices is equal to S with probability $P(H(N,S,1) = 0)$ or equal to $S-1$ with probability $P(H(N,S,1) = 1)$. If a chain whose block index is greater than $t-1$ is selected, it may contain a new block index of C' or any other new chain block indices greater than $t-1$. We approximate this probability by

considering only the probability that the chain drew C' during its last draw, i.e. $P(H(N,S,1)=0)$. Thus, we obtain:

$$\chi_2 \leq \chi_1\Bigg(P(H(N,S,1)=0)\Big(\sum_{j=0}^{S} P(H(N,S,S)=j)(P(H(N,S,1)=0))^j\Big) + P(H(N,S,1)=1)\Big(\sum_{j=0}^{S-1} P(H(N,S,S-1)=j)(P(H(N,S,1)=0))^j\Big)\Bigg)$$

Let $p_{2,k} = P(H(N,S,1)=0)\sum_{j=0}^{min(S,k)} P(H(N,S,k)=j)(P(H(N,S,1)=0))^j$. Then $\chi_2 \leq \chi_1\Big(P(H(N,S,1)=0)p_{2,S} + P(H(N,S,1)=1)p_{2,S-1}\Big)$. Taking into account the case where $k \geq N$, i.e. all chains have been drawn at least once (so the probability that C' has not been drawn is 0), we can apply the formula of $p_{i,k}$ iteratively until $i = \rho_{(N,S)}$. □

Theorem ?? $\forall N, \forall S \leq N$, $\exists \rho_{(N,S)} > 0$ such that $\chi_{\rho_{(N,S)}} < \epsilon$.

Proof. Direct from Lemma **??** □

C.7 Safety of the core

Definition C.10. *The core of a chain is composed of the new core validators of the last $\delta - 2$ consensus. Let $X = X^A + X^H$ be the number of validators in the core, where X^A is the number of malicious core validators and X^H the number of honest core validators. We respectively denote X_r^A and x_r^H the number of corrupt and honest new core validators at the r-th last consensus, with $X_r = X_r^A + X_r^H$.*

Lemma C.7 *The probability that the core of a chain is corrupted is given by*

$$p_{core} = (1-\tau/c)^{(1-\mu')(c-c_{(\delta-1)})} \tag{10}$$

Proof. New core validators are elected by sortition during each consensus. A validator can only be elected as a core validator during its first consensus. Furthermore, the list of new core validators is only determined at the end of the consensus execution. Thus, core validators can only join the core for $\delta-2$ consensus executions. The probability of a validator u being elected as a core validator is $B(w_u, \tau/w)$[**?**], where B is the binomial distribution, w is the total validation stake of the consensus committee and w_u is the stake of validator u. In SplitChain, all validators have the same validation stake, thus $w_u = 1$ and:

$$\begin{aligned} X &\sim \sum_{(\delta-2)}^{r=1}\sum_{c_r}^{i=1} B(1,\tau/c) \\ &\sim B(\sum_{(\delta-2)}^{r=1} c_r, \tau/c) \\ &\sim B(c - c_{(\delta-1)}, \tau/c) \end{aligned} \tag{11}$$

We consider the worst case where the adversary concentrates the corrupted validators of the consensus committee among the new validators of the last $\delta-2$ consensus. Following the same logic as with equation **??**, with a proportion of $\mu' = \mu(\delta-1)/((\delta-2))$ corrupted validators, we obtain $X^H \sim B((1-\mu')(c - c_{(\delta-1)}), \tau/c)$. Thus, the probability of corruption p_{core} of the core is:

$$\begin{aligned} p_{core} &= P(B((1-\mu')(c-c_{(\delta-1)}),\tau/c) = 0) \\ &= \binom{(1-\mu')(c-c_{(\delta-1)})}{0} (\tau/c)^0 (1-\tau/c)^{(1-\mu')(c-c_{(\delta-1)})-0} \\ &= (1-\tau/c)^{(1-\mu')(c-c_{(\delta-1)})} \end{aligned}$$

□

Theorem ?? For any security parameter $\epsilon \in (0,1)$, for an adaptive adversary with any delay $\delta \geq 3$, there exists a finite core size τ_{min} such that $\forall \tau \geq \tau_{min}$, the probability p_{core} that a core of τ validators is corrupted satisfies $p_{core} < \epsilon$, with $p_{core} = (1-\tau/c)^{(1-\mu')(c-c_{(\delta-1)})}$.

Proof. We know from Lemma **??** that $X^H \sim B((1-\mu')(c-c_{(\delta-1)}), \tau/c)$, with $\mu' = \mu((\delta-1))/((\delta-2))$. The expected proportion of honest core validators is $E(B((1-\mu')(c-c_{(\delta-1)}),\tau/c)) = \tau(1-\mu') + \tau(c_{(\delta-1)}/c)(\mu'-1)$. As μ' decreases when δ increases, with $\lim_{\delta\to\infty}\mu' = \mu$, $\forall\delta \geq 3$, $\mu < 1/3 \Rightarrow \mu' < 2/3$. If $\mu = 1/3$ and $\delta = 3$, then we have $E(B((1-\mu')(c-c_{(\delta-1)}),\tau/c)) = \tau/6$. As p_{core} decreases with the value of δ and τ, $\exists\tau_{min}$ such that $\forall\delta \geq 3$ and $\forall\tau \geq \tau_{min} > 0$, we have $p_{core} \leq \epsilon$, where $\epsilon < 0.5$ is the security parameter. □

D Experiments

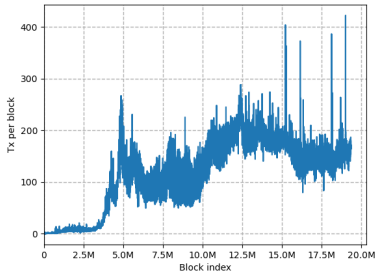


Fig. 6: Number of transactions per block (Ethereum)

To evaluate the efficiency of SplitChain's dynamic sharding compared to static sharding, we replay the entire transaction history of Ethereum from its

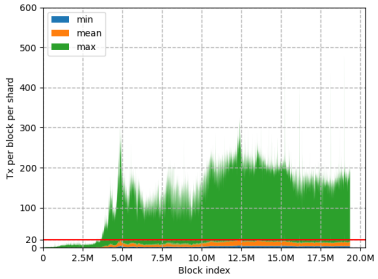
launch in July 2015 to block 19,353,052 in March 2024. This amounts to a total of more than 2.280 billion transactions. Blocks and transactions are extracted from a go-ethereum node using ethereum-etl, then stored and ordered in an SQLlite3 database. Our experiment is implemented in C++. Figure **??** displays the average number of transactions for each group of consecutive $1K$ blocks.

To assess the efficiency of transaction allocation for static sharding and SplitChain, we evaluate, for each $1K$ block segment, the minimum, average and maximum number of transactions processed by the chains in each of the two models. Figures **??** and **??** display the distribution of transactions for a static sharding solution of 32 and 256 chains. In both cases, the difference between the average and the maximum is very high: At block 12.5 billion, the maximum is 17 times higher than the average for 32 chains, and 131 times higher for 256 chains. In particular, we notice that the maximum load is almost identical in the two cases, indicating that although multiplying the number of chains by 8 reduces the average load on the chains, the distribution of transactions remains uneven.

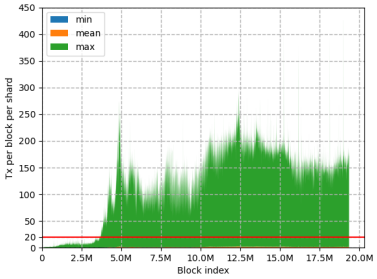
Figure **??** shows the variation of the number of chains in SplitChain. We have opted for a split threshold of 20 transactions and a merge threshold of 5 transactions per block. Having a low split threshold reduces computation time and the volume of information transmitted between validators within a chain, resulting in faster consensus and lower transaction-confirmation latency. Chains decide whether to split or merge every $1K$ blocks. We observe a strong correlation between the number of chains and the number of transactions. The number of chains used by SplitChain is also low, reaching 51 at most. Finally, Figure **??** shows the distribution of transactions in SplitChain. We notice that once a given total number of transactions has been reached (around 100 transactions per block), the average load quickly stabilizes at around 10 transactions per block, with a maximum load that is only 2.5 times higher at block 12.5 billion. Indeed, splitting chains efficiently spreads the load of the busiest chains, while creating far fewer chains than static sharding for the same label length. We conclude from this experiment that SplitChain's dynamic sharding better distributes transactions across different chains than static sharding.

Hotspot addresses. Ethereum's history includes hotspot accounts, i.e. accounts producing a very large number of transactions during one or several $1,000$-block segments. Given that a user account cannot be spread over multiple chains, a hotspot address causes repeated splits in the chain handling it, resulting in the creation of a multitude of unused chains that cannot be merged. The following method is used to detect hotspots and cancel a split operation.

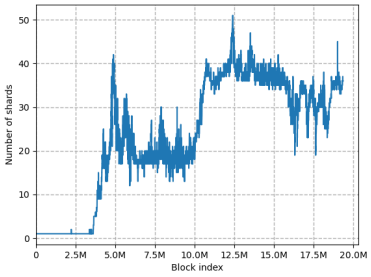
Let m be our hotspot-address detection parameter. Let's divide the account space of a chain C into 2^m subsets. Each subset corresponds to the account space of a descendant chain of C after m divisions. We say that C contains a hotspot if the same subset of C contains enough transactions to initiate a split. In that case, C does not split. For our experiment, we chose $m = 10$, which corresponds to 1024 subsets.



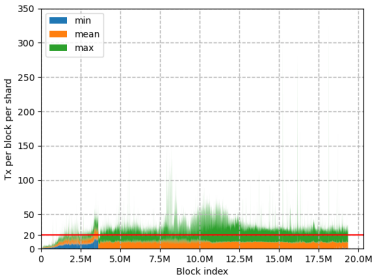
(a) Number of transactions per block per chain, 32-chains static sharding



(b) Number of transactions per block per chain, 256-chains static sharding



(c) Number of chains per 1K blocks segment, SplitChain



(d) Number of transactions per block per chain, SplitChain

Fig. 7: Efficiency of transaction distribution between chains.