Audit-Based Sharding for Public Blockchains

Kai Mast
University of Wisconsin-Madison

Charles Yu
Cornell University

Aaron Chao
Cornell University

Soumya Basu
Cornell University

Emin Gün Sirer
Cornell University

Technical Report

Abstract

Blockchains distribute data and execution across a public peer-to-peer network. This distribution renders them highly resilient against failures but also fundamentally limits their performance as they require every node to execute every transaction. As a result, systems such as Bitcoin or Ethereum support only about ten transactions per second on chain.

This paper presents *BitWeave*, a blockchain protocol that enables parallel transaction validation and serialization while maintaining the same safety and liveness guarantees provided by PoW protocols, such as Bitcoin, on the underlying blockchain. BitWeave partitions the system's workload across multiple distinct shards, each of which then executes transactions mostly independently, while allowing for serializable cross-shard transactions. The protocol relies on audit mechanisms to detect and punish misbehaving shards instead of complex committee selection schemes.

Evaluation shows that this scheme scales linearly with the number of shards As a result, BitWeave upholds Bitcoin's security and decentralization while scaling to thousands of transactions per second. We further demonstrate that performance does not degrade significantly during shard failures.

1 Introduction

Blockchains are a promising technology to enable decentralized applications. In particular, blockchains perform state machine replication across a public peer-to-peer network. This means they can support high-value applications, such as international payment processing [30], online auctions [38], and supply chain management [23], without the reliance on a trusted party.

However, in practice today, systems like Bitcoin or Ethereum are limited to tens of operations per second. The limited performance of these systems stems from the fact that they require all participants of the network to validate every transaction which significantly limits the overall throughput of the system. This makes it impossible to support the workload of any real-world application at scale.

We introduce *BitWeave*, a scalable transaction sharding protocol that is directly applicable to Bitcoin, Ethereum, and other public and private blockchain systems. At a high level, sharding allows breaking the collective workload of a system into smaller workloads that can be processed mostly independently. BitWeave implements sharding by building on the insight from Bitcoin-NG [12] that "mining" of a block in traditional Nakamoto consensus performs two tasks: leader election and transaction serialization, which are separable tasks. BitWeave leverages this to differentiate between protocol leaders, that perform consensus, and shard commanders, that serialize transactions.

The BitWeave protocol enables shards to execute mostly independently and relies on audit mechanisms to ensure correctness. The protocol distinguishes between the main chain, that mostly processes meta-information, such as who is responsible for which shard, and shard chains, that contain actual transaction information. Shards are not delegated to a trusted third party or a committee of nodes but are instead serialized by an *untrusted* node: the shard commander. Shard participants can rectify failure or misbehavior of shard commanders using the main chain. In the common case, this allows for high throughput, while safety is guaranteed during Byzantine failures.

To our knowledge, BitWeave is the first sound blockchain sharding protocol. BitWeave maintains security by avoiding dilution of mining power across shards, provides a sound incentive structure, and upholds decentralization by not giving large mining pools additional advantages. Instead of assigning a certain fraction of the mining power (or stake) to each shard, all mining power remains at the main chain. The number of information processed by the main chain is kept minimal, which allows nodes to participate in the consensus protocol virtually independently of their computing power. This, in turn, avoids centralization around a few powerful entities.

2 Background

We now give a high level overview of the problems BitWeave is addressing and how it compares to other approaches. Note, this section does not give a comprehensive overview over related work. We refer to Section 6 for that.

2.1 What are permissionless blockchains?

Permissionless blockchain systems are decentralized, replicated state machines with open membership. In general, these systems are comprised of a set of nodes, connected by a peer-to-peer network overlay, and operate in an environment without public key infrastructure. The goal of a blockchain is to facilitate the replication and ordering of state-transition operations in order to provide a globally consistent state.

Many permissionless blockchain systems implement some protocol of the following nature. First, there is a process LEADER that selects a designated node from the network of participating nodes. The designated node is awarded the right to propose state updates for some time called an epoch, whose duration is determined by a function EPOCH. The node performs these proposed state updates via a process ORDER in which the node sequences some set of state-transition operations and propagates the proposals to the rest of the network. In most systems, the leader is generally incentivized with some reward for performing ORDER. Depending on how the LEADER process is defined, situations may arise in which multiple protocol leaders are assigned for a given epoch, resulting in forks, or parallel histories of operation sequences. If the protocol produces forks, there must be a mechanism for deciding between conflicting forks, so that eventually the protocol converges back to a single history.

To make the preceding generalization concrete, we examine Bitcoin, which implements *Nakamoto Consensus*. In Nakamoto Consensus, LEADER and ORDER are facilitated through Proof-of-Work (PoW): Miners, or nodes competing for protocol leadership, pick a group of transactions from the network and create a block. To propose this block as an update to the blockchain, they compete to solve a cryptographic puzzle, whose solution designates the solver as a protocol leader. The solved block is broadcast to all other nodes in the network, and each node will accept it after successfully validating it. The block solver is entitled a monetary reward if it is globally accepted, and all miners repeat this process for the next process. EPOCH is implicitly maintained through a global difficulty parameter which is set so that the entire network of miners is likely to solve the puzzle, i.e., mine a block, only once in a specified interval. As a LEADER process, PoW introduces the possibility of *forks* when two miners propose competing blocks at the same time. To resolve forks, miners in Nakamoto consensus fork to build on and the fork that gets extended first "wins".

2.2 Why do current protocols not scale?

We now discuss the major bottlenecks of blockchains: *verification*, *execution*, and *communication*. Essentially, the LEADER and ORDER processes as described require massive replication of both data and computation. Thus, what the network can process as a whole is limited by the fact that every participant needs to process, forward, and execute all transactions.

First, transactions in blockchain systems differ significantly from those in conventional database systems. In blockchain protocols, each node maintains the local state in an authenticated data structure to be able to verify and process future blocks. In particular, blockchain nodes usually calculate and store some form of Merkle-tree of the state, and every block contains the root hash of the current state. These hash trees can be used both to verify blocks and to provide succinct proofs of some substate of the system. Executing transactions in such an authenticated manner requires more computation and storage. This is one of the reasons why systems such as Ethereum employ a limit on how much computational steps a block can contain ("gas limit"). Previous work has demonstrated that an improved storage engine can mitigate this bottleneck to some amount [33].

Second, blockchains rely on digital signatures to ensure the correctness and authenticity of messages. Intuitively, checking every transaction request and block generates a high computational workload as digital signatures are rather complex to verify. Increasing the number of transactions for some time interval thus significantly increases the burden for every node in the network to participate in the protocol.

Third, in order for every node to be able to process every block and transaction, all transactions and blocks must be propagated to the entire network. Intuitively, this creates a high network communication overhead. Blockchains usually execute across a geo-distributed peer-to-peer network. Here, larger state that needs to be synchronized will further increase the considerably high propagation latencies.

Even worse, scalability mechanisms may harm decentralization, a key promise of blockchains. Bigger block sizes raise the CPU and storage requirements for nodes participating in the network. This problem is especially salient for new nodes joining the network the need to verify *all* blocks in the chain before processing new transactions. As a result, only participants with strong hardware that are well-connected may participate in the protocol, causing a more centralized network layout. Further, merely increasing the block size in Nakamoto-based system do not necessarily improve throughput [13, 10]. While larger blocks can hold more transactions, they take longer to propagate through the network. Thus, the network is more likely to fork and, in turn, discard blocks. Similarly, increasing the frequency of newly mined blocks leads to a higher likelihood of forks as well.

2.3 Why is sharding blockchains hard?

Sharding aims to address all three scalability bottlenecks by breaking up the blockchain's workload and assigning it to a set of shards, each of which can operate mostly independently. Sharding then allows every participant of the protocol to process only a subset of all transactions depending on which shards they subscribe to. Ideally, this allows to linearly scale the throughput of the system *without* increasing the burden on any particular participant.

However, realizing a sound and effective sharding protocol faces significant challenges. Previous work usually implements sharding in the following way [39]. Some mechanism keeps track of a set of identities, e.g. by examining the last k miners of a PoW chain [20]. The protocol then assigns shard some subset of these nodes. Each shard then locally runs a consensus mechanism, such as PBFT [5], and a distributed transaction protocol, such as two-phase commit, handles cross-shard transactions. Finally, some scheme is in place to periodically "merge" the state of all shards. Drawbacks in existing sharding solutions can essentially be broken down into three categories: *reduced safety*, *loss of network decentralization*, and *lack of economic incentives*. If implemented incorrectly sharding may harm immutability, a core safety property of blockchains. Blockchains provide immutability by making it economically or computationally expensive for an adversary to undo transactions or tamper with the state. For example, Bitcoin would require controlling more than 25% of the mining power to modify the state [14]. A naive sharding solution would simply split all miners into k shards, which would reduce the safety of a shard by a factor $\frac{1}{k}$ compared to the system as a whole. Solutions such as Monoxide [40] address dilution of mining power by relying on large-scale mining facilities to concurrent process all shards. Mining a blockchain or a shard chain requires verifying and executing every single transaction of that chain. This concurrent mining scheme is thus not possible for most node in the network but only large scale actors, which, in turn, harms decentralization.

Finally, OmniLedger [21] is a safe solution for sharding but has some practical limitations. OmniLedger relies on provably-random assignment of nodes to shards to prevent colluding parties to be assigned to the same shard. The protocol randomly assigns each shard a quorum of nodes at the beginning of every epoch, which has two major drawbacks. First, whenever nodes are reassigned to a different shard, they need to update their local state which is computationally costly. Second, as the protocol requires the existence of numerous shards, each with their own replica set, transaction fees are potentially split between many participants, which makes it hard to build a sound incentive structure. Further, as node are assigned to a random shard, they might have no stake in that particular shard, which further disincentivizes them from participating in the protocol.

Part of the Ethereum 2.0 protocol is sharding mechanism similar to OmniLedger. Here each shard is assigned a large set of "notaries" which have a similar function as the per-shard quorum in OmniLedger. To the best of our knowledge, no formal specification of this sharding protocol exists yet, but we expect it to face similar challenges as OmniLedger. Further, it is unclear if Ethereum 2.0 will support serializable cross-shard transactions.

2.4 What is Bitcoin-NG?

While, in most protocols the LEADER and ORDER tasks are bundled together in one process, Bitcoin-NG [12], on which BitWeave builds, breaks down the process of mining in traditional Nakamoto consensus into its constituent processes to increase throughput. The Bitcoin-NG LEADER process proceeds as follows: Miners solve a PoW puzzle and broadcast a special block called a *keyblock* with the solution to the rest of the network, signaling their status as the protocol leader.

Once a miner has been elected leader, it continuously performs the ORDER process by grouping transactions into a sequence of *microblocks* until a new leader is elected. Miner here do not mine on the key block, but on the most recent microblock they have seen to maintain a serial order of blocks. In Bitcoin-NG, solely the network speed and how quickly the leading miner can sequence transactions limits the overall throughput.

While Bitcoin-NG improves throughput over the conventional Bitcoin protocol, it is still limited to the bandwidth of a single entity executing the ORDER-process. BitWeave addresses this limitation by accommodating multiple distinct microblock chains that each can be ordered by a distinct entity, which we discuss in the next section.

3 The BitWeave Protocol

BitWeave introduces a novel sharding protocol design allows for concurrent serialization of transactions without sacrificing security guarantees afforded by traditional Nakamoto consensus. It does so by allowing the network to self-organize into pools of nodes that concurrently process transactions. Since most blockchains today use account-based transaction models, we describe how BitWeave parallelizes transaction processing under this model. However, BitWeave can be easily applied to other transaction models as well, such as UTXO-based transactions.

BitWeave enables sharding by supporting multiple, concurrent microblock chains. Each shard is responsible for maintaining a subset of all accounts. A transaction may operate on account data that exists in multiple shards, and each shard in which it operates will process the transaction. To enable scalability, BitWeave is designed to minimize the number of shards included in a transaction and the amount of communication needed between shards. Figure 1 shows how BitWeave compares to existing protocols, Bitcoin and Bitcoin-NG, at a high level.

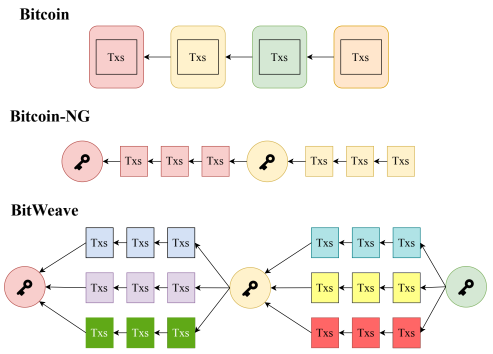


Figure 1: A high-level comparison of three protocols: Bitcoin, Bitcoin-NG, and BitWeave. Each color denotes a different participant in the system. The circular blocks with keys represent key blocks, and the "Tx" labeled blocks are transaction carrying blocks.

3.1 Assumptions and Attack Model

Like previous work [21, 12], BitWeave is designed to be resilient against a strong adversary that controls up to 25% of the network mining power (or stake). Honest nodes in this setup are entities that execute the protocol as prescribed. Malicious nodes may want to change the network's behavior to their advantage or break the network entirely. To achieve this, attackers may engage in arbitrary Byzantine behavior, such as issuing invalid transaction requests, delays or hiding of messages, and issuing conflicting microblocks. *Fraud* in this setting consists of a specific type of Byzantine behavior in which a shard commander intentionally issues an invalid block.

To guarantee liveness, BitWeave assumes an upper bound on network latencies and assumes the absence of long-lived network partitions. Bitcoin implicitly makes the same assumptions and does not guarantee safety in the presence of long-lived partitions. Additionally, like other decentralized systems, we assume that participants have a general interest to advance the protocol as long as they are financially incentivized to do so. So, similarly how it is unlikely to encounter an empty block in Bitcoin, even though miners are allowed to issue them, we expect shard commanders to publish a steady stream of microblocks to increase their revenue.

The BitWeave protocol expects that network participants have some incentive to validate the correctness of shards. This is a realistic assumption as shards followers usually have a stake in their currently running transactions, as well as the network itself. Existing blockchain systems have similar assumptions such as the existence of so-called "full nodes". An undetected failure might significantly lower the value of a cryptocurrency backed by the blockchain and thus hurt shard followers financially.

Finally, BitWeave leverages the same safety assumptions for epoch leaders as Bitcoin holds for its blocks. Eventually, an honest epoch leader will be elected and ensures correctness of the main chain. Thus, the protocol only has to accommodate Byzantine failure for shard commanders.

3.2 Consensus Abstraction

BitWeave provides a generalized mechanism that is independent of the underlying consensus protocol. This allows BitWeave to leverage advancements consensus protocols that are unrelated to sharding. Abstractly, blockchain consensus protocols agree on the order of an append-only log that contains transactions with some payload.

The underlying blockchain protocol must be able to support the semantics of a cryptocurrency and provide the notion of *epochs* to indicate the passage of time. A cryptocurrency is necessary for mechanisms that incentivized data pods to behave correctly. A new epoch can be indicated by the mining of a new block in Nakamoto-based systems or the admission of a new batch of transactions in a committee-based protocol.

3.3 Blockchain Structure

In general, a participant of a permissionless blockchain system may invoke certain system behaviors by sending a transaction request to the network. The network nodes that receive the request keep it in their local storage until they have verified it and included it on the chain or have discarded it for being invalid. In BitWeave, these transaction requests contain a source and target account and may contain an amount to be transferred between the two accounts, a function invocation or both. The issuer of the transaction, i.e. the owner of the source account, further signs the transaction so that other participants can verify the authenticity of the request.

BitWeave supports many concurrent shards, each of which serializes transactions by bundling them into *microblocks* and publishing them onto their own distinct chain. These microblock chains are periodically joined by a *keyblock*, which establishes an order between microblocks of different shards. BitWeave nodes leverage the ordering provided by keyblocks to establish a happens-before invariant in transaction processing – specifically, they ensure that all shards approve of a proposed transaction with a high certainty before it is committed. As a result, each shard in BitWeave establishes a total order among its microblocks, while there exists only a partial ordering between microblocks across different shards.

BitWeave allows different participants of the network to track only the transaction data of shards they follow. While the exact layout of the microblock varies depending on the underlying protocol, it always follows the following structure: The header contains meta-information and is cryptographically signed, and the payload includes transaction data. Unlike Bitcoin-NG, the same transaction might appear in the payload of multiple microblocks, representing different stages of the execution of the transaction. The payload of a microblock contains a sequence of transactions, each followed by a flag corresponding to the specific action regarding the transaction, such as COMMIT or RESERVE. In the following section, we differentiate between *shard chains*, which contain serialized transactions, and the *main chain*, which contains fraud proofs and delegation information. While the former is prone to fraud, the latter is protected by Proof-of-Work (or a similar mechanism). We formalize our definition of fraud in Section 3.7.

3.4 Roles

Participants in the BitWeave network can hold one or more the following roles: *miners*, *epoch leaders*, *shard commanders*, and *shard followers*.

Miners and Epoch leaders Epoch leaders decide which microblocks from the previous epoch are included on the main chain, delegate shard commandership and handle recovery after detecting shard misbehavior. Epoch leaders are selected from a pool of miners through some LEADER process such as Proof-of-Work or Proof-of-Stake, and publish a key-block to the blockchain to denote their leadership. Epoch leaders only maintain meta-information about the chain state – all regular transaction are handled by shard commanders and shard followers.

Shard followers Shard followers maintain state for a particular shard and processes the shard's microblock chain. Each shard follower thus maintains a subset of the system-wide chain state. Unlike other roles, the LEADER process does not select shard followers. Instead, any participant in the network may opt to follow a shard's chain and validate operations on it. When followers detect shard commander misbehavior, they report it by issuing a fraud proof to the network.

Shard commanders A shard commander is a particular shard follower that is appointed to process transactions related to a specific shard. Shard commanders perform the ORDER process for their shard by serializing transactions into microblocks on its shard until a new key block is mined. When a key block is mined, the epoch leader names one shard commander per shard through their public key.

3.5 Transaction Processing Overview

Designing a transaction protocol for a sharded blockchain reduces to two fundamental challenges: dividing work among shards and ensuring atomicity in cross-shard transactions. At a high level, BitWeave implements a replicated state machine and a sound transaction processing mechanism is necessary so that the system always remains in a consistent state. Transactions in BitWeave are thus required to be serializable.

We first describe how a system's transaction workload is mapped to shards. Transactions in BitWeave are composed of a non-empty set of operations, where each operation represents a modification to data and applies exactly to one account. Additionally, we assume that there exists a function that derives the set of operations from a transaction. For example, a money transfer operation consists of a set of decrements on the source account(s) and a set of increments on the destination account(s). Because of the sharded nature of the blockchain, these operations may take place on different shards.

Each shard in BitWeave is responsible for maintaining a subset of all accounts and for processing operations affecting that those accounts. The protocol further defines a consistent hash function [18] that provides a mapping from accounts to shards. Since transactions generally operate on more than one account, it is often the case that several shards are involved in processing a transaction.

The second challenge of sharding is ensuring that all transactions are executed atomically – all shards participating in a transaction's execution must unanimously decide to whether or not to execute the transaction, and that execution needs to happen in lockstep across all participating shards. For single-shard transactions, this is trivial because there is a total ordering of transactions within an individual microblock chain. Therefore, single-shard transactions are simply included on their respective chains, just as in Bitcoin-NG.

3.5.1 Transaction Processing Primitives

For cross-shard transactions, BitWeave relies on a variant of the two-phase commit protocol. In phase one, participating shards implicitly signal to the rest of the system that they can perform their share of operations for a given transaction by issuing *reservations*. The shard maintaining the source account of the transaction further certifies that the digital signature associated with the transaction request is valid. Once reservations have been issued by each involved shard and seen by all other shards, the transaction is applied to the involved shards via a *commit*. Shards release all issued reservations once they learn of the commits on the other shards. Section 3.7 describes how this scheme is extended to handle Byzantine behavior.

Reservations Reservations are a primitive that ensures serializability of concurrent transactions. Reservations allow for greater concurrency over traditional locking by allowing multiple operations to hold locks on an object in some cases. For example, more than one transaction can spend from the same account as long as the account's balance is sufficient. Each reservation is bound to one specific operation that is intended to be applied to a single data object. Increasing concurrency by composing reservations is a crucial feature in BitWeave as transactions have high latency and traditional locking would limit throughput significantly.

More concretely, reservations contain a requirement for the input state of an account, as well as the proposed modification to that account. Thus, they can be expressed as a pair $(a; b)$ of a precondition a and a post-condition b and applies to one specific shard. As an example, money transfers require two different predicates: $(balance \geq i; balance \leftarrow balance - i)$ on the spending account and $(; balance \leftarrow balance + i)$ on the receiving account, where i is the amount transferred. This mechanism is extensible to arbitrary predicates, which enables BitWeave to execute generalized transactions.

Nodes maintain a *reservation set*, which is a registry of all held reservations, as part of a shard's state. Shard commanders generate a Merkle-tree containing the hashes of all reservations and include the tree's root in each microblock-header. They apply the same scheme to include the current shard state in the microblock. Once a shard's chain admits a new reservation, the reservation set is updated accordingly.

To admit a new reservation to a shard, it must be checked both against the reservation set as well as the stable state of the shard since it must remain valid in all possible combinations of commits and aborts. Checking a reservation against the exponential number of states it must remain valid in is infeasible, nodes maintain an interval of possible values. For the case of account balances, transactions then are validated against the least possible account balance as well as the highest possible account balance to ensure the reservation can be applied in all cases.

Commits and Aborts After a shard has included a transaction's reservation in its chain, it will eventually either include a corresponding ABORT or COMMIT entry. Commits denote that all affected shards have included the required reservations on their chain. Once a commit has been included in the chain, shards will release the associated reservations and modify the chain state accordingly. If a transaction does not acquire all required reservations in time, shards issue ABORT operations for the transaction. Once the chain includes the abort, shards release all associated reservations without modifying the chain state.

3.5.2 Efficient Cross-Shard Communication

BitWeave enables lightweight message passing between shards by only forwarding relevant reservations from one shard to another. This allows nodes in the system to verify a partial view of the total state: Nodes solely follow the full chains of shards they are interested in and rely on messages from other shards to reason about cross-shard transactions.

The protocol supports two different kinds of microblocks: *message blocks* and *transaction blocks*. A message block concisely summarizes the results from the previous epoch for other shards. Transaction blocks admit new transactions to the shard's chain or commit currently pending transactions. Shard commanders must publish a message block at the beginning of each epoch.

The message block contains a hash of the new state and multiple payloads, each containing a set of messages for a specific shard. A message consists of the transaction's identifier as well as a Merkle-proof certifying the reservation was acquired.

To efficiently apply this scheme, nodes in BitWeave connect to multiple relay networks. Namely, there is one main network that propagates block headers and key blocks, and multiple shard-specific networks exist that propagate transaction data. To implement such a scheme, nodes advertise their subscriptions to specific shard upon connecting to other peers. Nodes then ensure they are connected to a sufficiently large number of peers for each network.

3.6 Transaction Fees and Miner Rewards

A core component of Bitcoin are built-in incentive mechanisms for network participants, namely rewards for mining new blocks and transaction fees for validating transactions. While the reward scheme for newly mined key blocks can directly be derived from Bitcoin's mechanism, BitWeave transaction fees require a more complex scheme due to the existence of shards and microblocks. BitWeave treats transaction fees for each shard independently to reduce the amount of communication required during transaction execution. Transaction requests thus specify fees per shard incentivizing users to spread load, as more congested shards will have higher fees.

BitWeave splits fees three ways for each operation on a shard's chain, to maximize throughput. The intuition behind this is merely an extension of the proof provided by Eyal et al. [12]. Their work showed that fees should be split 40/60 between two consecutive epoch leaders to incentive the current leader to include as many blocks as possible from the previous epoch. BitWeave extends this scheme by allowing the current epoch leader and the current shard commander to split the first 40% of the fee, the ratio of which is determined between the specific shard commander and the epoch leader during shard delegation.

The main chain keeps track of a small set of accounts for actors involved in leader election and transaction processing. For nodes to be able to become miners or shard commanders, they need to create a globally viewable account. For miners, this account is solely used for rewards, while shard commanders need to put up a deposit to start their tenure as commander. To do this, they either need to collect funds by mining blocks or by transferring funds from a shard's account. Similarly, shard followers need to maintain funds on the main chain to be able to issue fraud proofs (Section 3.7.2) and availability wagers (Section 3.7.4).

Miners do not parse the entire content of shard blocks to extract fees but merely the block header, which contains the accumulated fees. Thus, miners rely on shard followers to validate the correctness of the fees specified in the shard block headers. To make this scheme secure, funds created from fees are locked until the associated transaction is validated.

3.7 Fault-Tolerant Transaction Processing

To address Byzantine failures, such as malicious nodes or network partitions, the protocol assigns an overall timebound to

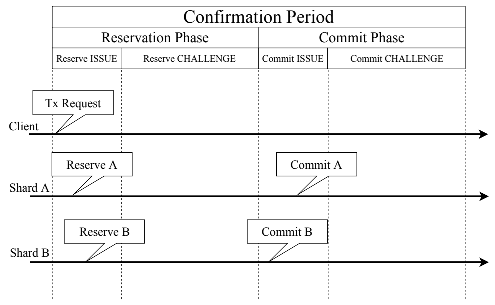


Figure 2: Lifetime of a cross-shard transaction: The transaction is not considered finalized until all reservations and commits have been included and validated on all shard chains.

each transaction, the *confirmation period*, which allows for ample time to verify a transaction's correctness before it is applied to its corresponding shards' state. A transaction is thus atomic because it does not modify state until it passes through the confirmation period on all shards and all locked resources are released if the transaction does not get confirmed in time or aborts.

Figure 2 sketches how the confirmation period is broken down for a cross-shard transaction. As the protocol is an extension of two-phase commit, we can separate the confirmation period into a *reservation phase*, in which the required resources for the transaction are locked via reservations, and a *commit phase*, in which the transaction is confirmed or aborted. To account for Byzantine behavior, each phase proceeds in two stages. The ISSUE stage, where shard commanders issue acknowledgments on-chain to signify that shard-specific operations were performed. Then, a CHALLENGE stage, where Byzantine behavior in the ISSUE stage is amended through a fraud proof and rollback mechanism (Section 3.7.2) carried out by shard followers. To ensure that shards are coordinated on which stage of the confirmation period is occurring, each stage is measured in a certain number of epochs from the key-block height at which the transaction was created by the client, the length of which is mandated by the protocol.

3.7.1 Detecting Fraud

BitWeave depends on shard followers to aid in validating of transactions and detecting fraud. Reservations and commits cannot be immediately assumed to be valid, as malicious behavior during the ISSUE stage of each phase is resolved during its corresponding CHALLENGE stage through fraud proofs submitted by shard followers. Followers are motivated to participate in this behavior through built-in financial incentives. Correctness can be maintained by the presence of a few honest shard followers and does not require a majority of honest shard participants.

The length of the CHALLENGE stage must be set to a sufficiently large number of epochs, so that an issued reservation, commit, or abort guaranteed to be correct after its CHALLENGE stage has passed. Instantiations of the BitWeave protocol thus set the length of this stage such that it is virtually impossible for a sequence of malicious epoch leaders to be active during their entire duration and to tolerate network propagation latencies. To determine the length of the CHALLENGE stage, the protocol sets time bounds for the propagation delay t_p and the validation delay t_v, which is the period it takes for at least one honest epoch leader to be available. The protocol then sets length of the CHALLENGE stage as a function of these parameters, where $t_{challenge} = t_v + 2 * t_p$. Additionally, it should hold that the ISSUE stage is at least as long as t_p, to avoid unnecessary aborts.

A concrete implementation, based on Bitcoin-NG, sets those values to $t_p = 3$ and $t_v = 10$. We can show that for a validation delay of ten epochs we show in the probability that at least one honest epoch leader exists is very high. Assuming, like previous work, that at most 25% of the mining power (or stake) is controlled by malicious parties, the following holds.

$$\begin{aligned} &Pr[\text{“\# honest leaders} > 0\text{”}] \\ =\,&1 - Pr[\text{“\# honest leaders} = 0\text{”}] \\ =\,&1 - 0.25^k \approx 1 - 9.54 * 10^{-7} \approx 1 \end{aligned}$$

This probability is lower than the probability of the chain's last k blocks being rewritten by a malicious actor, based on the proof in the original Bitcoin paper. Bitcoin, like BitWeave, assumes a tight network propagation bound to ensure timely convergence of the chain.

3.7.2 Fraud Proofs

At a high level, *fraud proofs* are messages containing references to discrepancies in a shard's chain. The protocol differentiates between three different kinds of fraud proofs, each relying on a set of Merkle proofs to concisely demonstrate their correctness. First, a fraud proof is raised if a shard block is inconsistent with its header. This can be the case if any of the Merkle-roots are not consistent with the state stored in the block's body. Second, if a transaction block contains a reservation, commit, or abort that is not consistent with the shard's state, a fraud proof is raised. This can be demonstrated by making the fraud proof contain a snapshot of the involved object(s) and reservation(s). Third, a fraud proof is raised if a message block contains an invalid message, or misses a message. The former can be shown by demonstrating that the Merkle proof associated with the message is incorrect, while the latter is shown by pointing to a commit of the previous epoch(s) that does not have a corresponding message.

Fraud-proofs are submitted to the epoch leader, and they enable the epoch leader to correct a shard state by reverting the history of the shard to the state right before the conflicting block, thus nullifying the fraudulent transaction. Having epoch leaders process fraud proofs ensures that the malicious behavior is rectified on-chain and ensures that every party will see the correction, preventing nodes from operating on an invalid shard state.

Fraud proofs, once included in the main chain, override the shards state. Depending on the kind of fraud, either a specific block is invalidated, a certain operation is undone, or the contents of a message are modified. The previously described CHALLENGE stages ensure that rollbacks do not affect transactions that are considered committed and valid.

3.7.3 Incentivizing Fraud-Finding Behavior

BitWeave's incentive mechanism is designed to reward fraud finders and punish misbehaving shard commanders. Nodes pay a deposit to become a shard commander, which is withheld in case they misbehave. In the event that a shard follower detects misbehavior and successfully submits a fraud proof, the misbehaving commander's deposit is forfeited and collected by the follower.

Further, BitWeave employs two mechanisms to ensure that rewards are not stolen from honest validators. First, the system relies on cryptographic commitments to prevent followers from sending fraud proofs in plaintext. Followers issue commitments in the form of a cryptographic hash of the fraud proof, and not the proof itself. Once the main chain includes a commitment, followers can reveal the fraud proof's content to collect the owed reward unless an earlier fraud proof has already collected the reward.

Like regular transactions, fraud proofs include a fee to prevent denial-of-service attacks. Fees for fraud proofs render it infeasible for an adversary to issue many invalid fraud proofs. Because the fraud proofs rely on cryptographic commitments, the epoch leader cannot distinguish between valid and invalid fraud proofs at first; however, it can detect if the issuing party has enough funds to pay for the fraud proof transaction. Epoch leaders include fraud proofs independent of their validity and collect the associated transaction fee to limit the number of invalid fraud proofs that can be issued.

3.7.4 Ensuring Shard Availability

Similar to how fraud proofs guarantee safety in BitWeave, the protocol relies on probabilistic sampling and availability wagers ensure liveness.

Honest miners rely on sampling to ensure they only include shard blocks on the main chain, for a which the payload is available. First, shard commanders encode their blocks using Reed-Solomon error correcting codes, so that parts of a shard block's payload can be sampled efficiently. Then, instead of parsing the entirety of every block, they sample a small fraction of it from the network and reject all blocks where that sample is unavailable. Al-Bassam et al. [2] demonstrated that in this setting querying as little as 1% of a blocks content is sufficient to tell with very high probability if it is available or not, and we refer to their work for a more detailed description of this sampling scheme.

If a malicious miner includes an unavailable block, followers issue an *availability wager* to the main chain requesting a specific block's payload to be published. A shard follower that issues an availability wager must attach funds that represent their confidence that a commander is misbehaving and multiple wagers from different shard followers can be attached to the same block. If the commander does not reveal the block after a set time, the challenging follower is rewarded the wager amount, half taken form the commanders deposit, the rest taken from the miner that included the unavailable block.

The protocol requires followers to pool funds to ensure no unnecessary wagers are processed and to provide a shard availability heuristic for followers. Upon detecting a potentially unavailable block, a shard follower issues a wager with some funds attached. Wagers must reach a certain threshold of funds before epoch leaders may include them in the chain. When the wager propagates through the network, other followers can either opt to attach more funds to the wager or reveal the block payload to the network if they have access to it. Eventually, the wager will either collect enough funds to be included in a key block, or it expires because the block was made available.

If the commander reveals the block in time, the follower's wager instead is burned and the protocol proceeds without change, which ensures that shard commanders do not profit from withholding blocks. Thus, a rational shard commander will always make their block payloads available as soon as possible, to ensure their block headers are included in the main chain.

3.7.5 Adaptive Confirmation Periods

The guarantee that there exists at least one honest epoch leader during each CHALLENGE stage, is not sufficient for all cases. In particular, a single key block might not be able to hold all fraud proofs and availability wagers currently in the network. This problem is exacerbated by the fact that BitWeave makes no assumptions about the correctness of the shard commander. In the worst case, a single malicious actor could be in control of many shards, which would result in the main chain not including fraud proofs before a transaction's contest or validation stage expires.

The BitWeave protocol is designed to extend a transaction's confirmation period in the case of fraud to allow for enough time to reconcile the chain state using *global challenge extensions*. Key blocks contain a flag that indicates whether all fraud proofs have been processed. If this flag is set, a global challenge extensions is issues which extends the CHALLENGE stages of all pending transactions by the validation delay t_v. The latter ensures another honest leader is around in time to process the remaining fraud proofs. Availability wagers similarly extend the CHALLENGE stages of the affected shard to allow time for the newly revealed block to propagate or for the chain to roll back. This means that an excess of fraud proofs will stall the overall throughput of the chain to ensure correctness, but ensures that safety of the protocol is always maintained.

Similarly, BitWeave accommodates shards with overloaded workloads with *commit extensions*, which lengthe the commit-ISSUE stage for currently pending transactions of that shard. In other words, followers of a shard do not count an epoch towards the commit-ISSUE stage of a transaction if there are still transactions to be committed on the shard or if a shard's epoch is empty, i.e., no message block is published for that epoch. This means the confirmation period of all pending transactions is extended to ensure they're is ample time to validate all commit and abort messages.

3.8 Correctness

We provide a proof sketch demonstrating that the BitWeave protocol upholds safety and liveness, assuming the assumptions of Section 3.1 are not violated and assuming the underlying consensus protocol behaves as specified in Section 3.2. In particular, we assume that there is a known bound on the network latency t_p and that the consensus protocol has at least one honest leader within the validation delay t_v. Then the CHALLENGE period is set to at least $2 * t_p + t_v$. Additionally, we assume each shard has at least one honest follower and the underlying consensus protocol does not violate safety or liveness.

Our proofs for safety and liveness rely on the following two lemmas:

Lemma 1: The number of challenges to a particular transaction, in the form of potential availability wagers, fraud proofs, global challenge extensions, and shard commit extensions, is guaranteed to be finite.

Proof. We first show that only a finite number of availability wagers can affect a particular transaction. Availability wagers can only affect a transaction at the ISSUE stage, which is a fixed number of epochs. Since epochs are finite, the number of blocks contained in the ISSUE stage is finite. A single transaction has at most two issue phases, at most one message per shard and will be involved in a finite number of shards. Thus, the number of blocks involved in a shard's ISSUE stage is finite, which means that the number of potential availability wagers involving this transaction is guaranteed to be finite.

Recall that global challenge extensions are issued when the main chain receives too many fraud proofs, and that shard commit extensions are issued when the number of commits to be processed exceeds the shard's capabilities. Global challenge extensions and shard commit extensions merely extend the current epoch and do not allow any new transactions. Since shards (and the main chain) are both assumed to eventually have an honest commander (or leader) who will continue processing, the number of global challenge extensions and shard commit extensions is finite. We see that only a finite number of fraud proofs that can be issued as all fraud proofs must be issued in the global challenge and shard commit periods.

Lemma 2: Any given shard will eventually accept new transactions.

Proof. BitWeave incentivizes shards to accept new transactions through the use of fees. Shards are financially incentivized to include transactions in their chain in order to collect their fees as revenue. As there is no base block reward for microblocks, if shard commanders fail to include any new transactions, they will fail to make any profit from their work. Therefore, any shard whose shard commander is rational and honest will accept new transactions.

A rational epoch leader is incentivized to replace a slow or non-responsive shard commander. If a shard does not process transactions, the leader that appointed the shard commander loses revenue. As a result, any rational epoch leader will be incentivized to pick responsive shard commanders, and replace unresponsive ones. A rational epoch leader will eventually be chosen due to the assumptions made on the underlying consensus protocol.

Combining those two facts above, we see that eventually a rational epoch leader will be chosen and such a leader will ensure that shard commanders are rational and honest. Consequently, any given shard will eventually accept new transactions.

3.8.1 Safety

BitWeave guarantees that (1) particular shards do not violate consistency and that (2) cross-shard transactions are atomic and consistent. The concrete consistency constraint is defined by the particular application. E.g., for cryptocurrencies BitWeave guarantees that account balances are always non-negative.

While a particular shard commander might misbehave, such misbehavior will be overwritten by fraud proofs on the main chain before the contest period has passed. As described before, a shard's stable state is guaranteed to be safe, while the pending state might exhibit inconsistencies. That means, only after the contest period has expired can clients expect the outcome of a transaction to be consistent and immutable. Transactions that are not yet fully executed or validated, are considered internal states of the protocol and should not be propagated to the end user.

Proof. To prove safety, we first show that all cross-shard communication happens within a particular time interval that is pre-set by the protocol. Then, we show that any particular shard, given the right messages and waiting the appropriate period, will not violate consistency and will agree on whether the transaction should commit.

A transaction first starts by reserving the objects that it will modify. A valid reservation must be included before the beginning of associated reserve-CHALLENGE stage, and a CHALLENGE stage is at least $2 * t_p + t_v$ epochs long. Shards must include a message block at the beginning of every epoch. If they do not, the particular epoch does not count towards a CHALLENGE stage. It follows from Lemma 2 that the number of epochs without message blocks is finite. Note that, if the next message block after a reservation does not include a message for that reservation, the associated fraud proof serves as a message instead. We thus see that for every reservation, a message is generated at least $2 * t_p + t_v$ epochs before the end of the associated CHALLENGE stage.

After the reservations are made, the transaction is committed or aborted at the corresponding commit-ISSUE phase. We now show that messages sent between shards are guaranteed to arrive before the corresponding commit-ISSUE phase begins. Shard followers and commanders parse the main chain for message blocks. Message block headers contain the hash and shard identifier for each of its payload. As a result shard followers, and leaders of other shards can always identify the existence of a message relevant to them. They can then request the message or resort to an availability wager if it is not available. As with transaction blocks, the availability wager extends the challenge period ensuring that it will arrive in time.

Thus far, we have shown that all reservations will be sent and received by all shards before the corresponding commit-ISSUE phase. This proves that all cross-shard transactions are atomic. We now must show that single shard and cross shard transactions do not violate consistency.

In particular, we show that for every operation on a shard chain, there exists a finite value k such that once the entry is buried k key blocks deep, it is guaranteed to be consistent. Recall that, every operation, e.g., the commit of a transaction, is recorded in a shard's microblock. A microblock is only considered part of the chain after it, or one of its successors, has been referenced by a key block. Shard followers track the main chain for block headers. Additionally, they parse all availability wagers and fraud proofs and adjust their state if they affect shards they follow.

For every operation part of some transaction T, k is at least $2 * t_p + t_v$, the minimum length of a regular CHALLENGE stage. If there is a header for a shard block and a shard follower has not received its payload after t_p, they issue an availability wager. The availability wager will then be propagated to the network in at most t_p. If a valid availability wager extends T's confirmation period by $a = 3 * t_p + t_v$, the length of the CHALLENGE stage plus an additional round trip time to allow the shard to propagate the block's payload, we also increase k by a. It follows from Lemma 1 that the number of the possible availability wager extensions of this form is finite.

Once a shard follower has received a microblock's payload, they verify it. When they receive the payload, there is at least t_p of the contest period remaining. Thus, if the follower finds inconsistencies, sufficient time remains to generate a fraud proof, to send it to the main chain, and for the main chain to propagate it to all participants of the protocol. A fraud proof, invalidates the affected entry(ies). Invalidated entries are always consistent with the shard's state as they do not modify the shard's state.

This completes the proof for safety.

3.8.2 Liveness

BitWeave guarantees that it will eventually decide to either commit or reject a transaction (*progress*) and that it will not reject all transactions (*non-triviality*).

Proof. We see that transactions are finalized and either committed or rejected within a finite amount of time by applying Lemmas 1 and 2. As there is only a finite number of possible extensions to a transaction and shard do not generate empty epochs indefinitely, all parts of a transaction will eventually be processed by each involved shard.

To prove nontriviality, we apply Lemma 2. Shard commanders that do not generate empty epochs, either include commit and abort messages for currently pending transactions or reservations for new transactions. Assuming the number of pending transactions is finite, shard commanders will eventually accept new transactions.

4 Case Studies

BitWeave applies to different consensus protocols and data models, the latter of which we describe in this section. Consensus protocols, both permissioned [5] and permissionless [30], that provide the abstraction of leader election can implement BitWeave's main chain. Because adapting BitWeave to different consensus protocols is trivial, this section merely discusses the data model of two popular systems based on Nakamoto consensus: Bitcoin and Ethereum.

4.1 Applying BitWeave to Ethereum

Ethereum introduced the notion of smart contracts, which allow for arbitrary programs to be executed as part of a blockchain protocol. Smart contracts can be viewed as a form of stored procedures, where each invocation results in the execution of a transaction. Usually, smart contracts are compiled from a high-level language to some form of assembly that can execute in a lightweight environment, e.g. the Ethereum Virtual Machine (EVM).

BitWeave already supports Ethereum's account model and can be extended to support smart contract execution. Clients prepare such transactions by executing smart contracts on their local state. From this, they derive a set of reads and writes that are mapped to BitWeave operations. While pessimistic locking might be more efficient in a geo-distributed setting, OCC keeps complexity of the protocol low and still performs well in the absence of high contention around certain data objects. Further, note that while end-to-end latency in BitWeave is high, the latency between client execution and reservation at the shard can be kept fairly low.

We sketch how this scheme works using ERC-20 [15] token transfers as an example. ERC-20 is a standard that allows developers to implement their transferable token using the Ethereum blockchain. In essence, these token can be transferred between users and exchanged against Ethereum's currency ether. ERC-20 tokens are implemented by a single smart contract that tracks a mapping from tokens to users. To buy tokens from another party, one needs to transfer funds to the account of that party. The remote party then invokes the ERC-20 smart contract to transfer the requested tokens. This three-way transfer must execute atomically to ensure nobody loses their ether or tokens without compensation during the process.

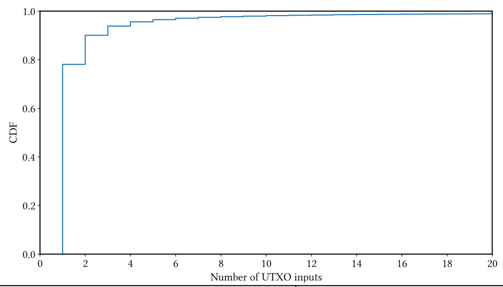
A token transfer between two parties, Alice and Bob, is implemented using a transaction that is applied to the two parties' accounts as well as the token's smart contract. The transfer of ether is implemented as before, where a reservation locks funds on Alice's and another reservation ensures that Bob's account exists. This transaction is then extended to apply another reservation to the smart contract that locks Bob's token. Once all three reservations are applied the transaction can commit, and the respective account and smart contract states are updated.

4.2 Applying BitWeave to Bitcoin

Thus far we have described BitWeave under the assumption of an account-based transaction model. Now we show that it is easy to extend BitWeave to accommodate an *Unspent Transaction Output (UTXO)* based transaction model, the same transaction model used in Bitcoin.

The UTXO Transaction Model The UTXO model represents an entity's wallet as a set of UTXOs, each with its public identifier. This model significantly reduces the complexity of the data model that transactions execute on, but prohibit storing custom state and procedures as part of the accounts. Platforms that are focused on monetary transactions, such as Bitcoin or ZCash, often rely on the UTXO model. Transactions in the UTXO model work similarly to a voucher system in which some input vouchers are exchanged for new vouchers of the same or lesser value. More concrete, transactions consume UTXOs (the unspent outputs of a previous transaction) and produce new UTXOs.

While in the account model data is sharded by account-id, in the UTXO model we can directly shard by transaction outputs to keep the number of shards involved in the transaction low. Therefore, a new transaction t with input UTXOs $u_1, u_2, ..., u_n$ is assigned to the shards responsible for the transactions that produced its inputs. To keep the number of shards involved in a transaction low, we then pick the identifiers of the transaction outputs such that they map to shards that contain one or more of the transaction's input.



Quantifier	Sep 2018 - Mar 2019
Number of Transactions	49,413,279
Mean Number of Inputs/Tx	2.248
Median Number of Inputs/Tx	1.0
Standard Deviation.	11.083

Figure 3: Statistics of UTXO inputs to Bitcoin transactions between September 1, 2018, to March 1, 2019. Most Bitcoin transactions only have one input.

BTC Transaction Survey We surveyed the BTC transaction history for a snapshot of six months, from September 1, 2018, to March 1, 2019, which includes a total of 49,413,279 transactions. Figure 3 shows this data in more detail, from which we can infer that the majority of recent BTC transactions (almost 80%) take in only 1 input, and the mean number of inputs to a transaction from the time series surveyed is 2.248. Therefore, given this empirical data and the sharding approach proposed above, the number of shards involved for an average BitWeave-BTC transaction is expected to be less than 3.248 shards (Assuming an average transaction's inputs are all from different shards and are unique from the transaction's output shard).

5 Experimental Evaluation

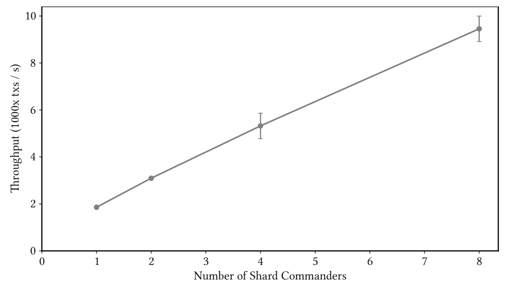


Figure 4: Scalability of BitWeave with increasing number of shard commanders

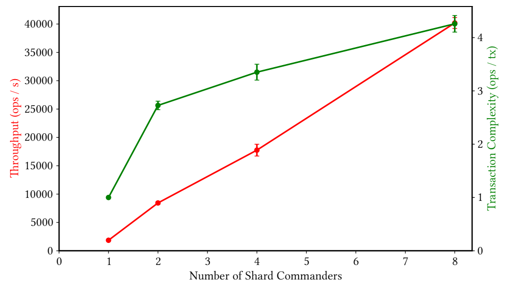


Figure 5: Total rate of operations (prepare, commit, abort) with increasing number of shard commanders

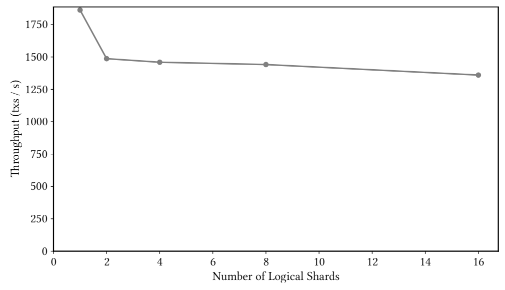


Figure 6: Overhead generated by sharding: We increase the number of logical shards while keeping the number of shard commanders constant

To have a realistic assessment we simulate a geo-distributed blockchain network for all experiments. All nodes are connected to a relay network, similar to how blockchain networks are connected through relays such as FIBRE [7]. The network simulates four different geographic regions with up to 100ms latencies between them. We run 200 miners with equal mining power ensure that a realistic number of forks are generated. Note, that current blockchain systems have usually about 10 to 20 mining entities with enough mining power to compete in the consensus protocol [16].

Each experimental run executes for about two hours to accommodate BitWeave's long living transactions. We then evaluate a period of 30 epochs in the middle of the run so that the measurement contains a mixture of reservations, commits, and aborts.

To assess the scalability of BitWeave, we configure a test setup that is bottlenecked on the processing power of the shard commanders to assess this. For each shard commander we allocate a distinct x1e.xlarge instance on Amazon EC2, which has 4 virtual CPU cores and 122 Gigabytes of RAM. The large memory requirement is not a limitation of the protocol, but a shortcoming of the prototype implementation which keeps all pending transactions in RAM.

In this experimental setup, we increase the number of logical shards and the rate at which new transactions arrive in the network with the number of physical shard commanders. In particular, we double the number of shard commanders at every configuration. For each configuration we ran three iterations of the same experiment.Additionally, we set the number of logical shards in the experiment equal to the number of shard commanders to get a fair comparison as each logical shard introduces more work for processing and executing transactions.

An excess of transaction requests creates additional verification work for transactions that will not be processed in time, while a low number of transaction requests results in the throughput of the system not being fully utilized. We thus set the rate of incoming transactions such that the CPUs are fully utilized on each shard. Clients issue a steady sequence of transactions sending money from the client's account to some other uniformly-sampled account. We increase the workload with the number of shards by increasing the number of client machines issuing requests. This ensure that clients are not the bottleneck.

5.1 How well does BitWeave's overall throughput scale?

Figure 4 shows how the throughput of the blockchain network as a function of the number of shard commanders. We evaluate the network with up to 8 shards, where the network can process over $9,000$ transactions per second. In particular, throughput scales linearly at a constant rate of about 1.6. Note, that executing BitWeave with a single shard is equivalent to the behavior of the Bitcoin-NG protocol.

From these results, we can conclude that assuming the existence of an efficient relay network, BitWeave's throughput is solely limited by the processing power of the shard commanders. More concretely, our experiments revealed that the major bottleneck occurs when transactions are issued and have to be verified by nodes they execute on. While there is theoretical upper bound for the amount of scalability possible, when verifying block headers generates a significant overhead, but we expect this limit to be far from reach for current configurations.

Further, the computing power of the machines running the shard commanders is rather modest and more efficient machines would probably yield even better overall throughput. Finally, the network bandwidth utilized by any blockchain node in any configuration was strictly below 20 Mbit/s. Thus, BitWeave can scale to support realistic workloads in a real-life environment.

5.2 How does sharding affect the transaction footprint?

Figure 5 demonstrates how transaction complexity, i.e., the number of operations involved in a single transaction, increases with a higher number of shards. If the number of shards is increased, more transactions execute across multiple shards. A cross-shard transaction consists of four operations (two reservations and two commits) and potentially more if a transaction needs to be aborted and resubmitted. Because of this, the transaction complexity stabilizes at slightly above four operations, while the number of total operations keeps doubling when the number of shards doubles.

5.3 What is the overhead generated by cross-shard messages?

To determine the overhead of cross-shard messages on the throughput, we conduct a microbenchmark where a single shard commanders executes a varying number of shards. Figure 6 shows the result. While more shards allow for more concurrency, they also generate more metadata on the blockchain. This overhead is especially salient when moving from one to two shards. After that, the penalty stays roughly constant as all transactions touch at most two shards.

5.4 How well does the protocol handle failures?

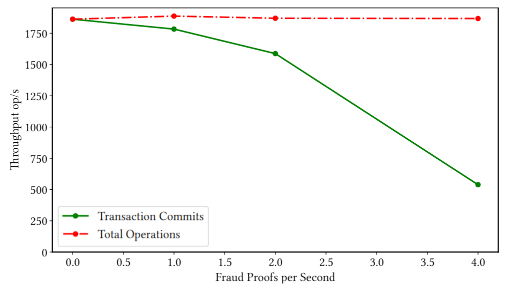


Figure 7: Performance with regard to fraud proofs. We run a single shard and gradually increase the frequency of transaction blocks that are invalidated.

We evaluated a microbenchmark which simulates shard failures to determine the overhead introduced by applying fraud proofs to the chain. In this experiment, we keep the client workload constant while introducing a steady sequence of fraud proofs into the network and observe how they affect performance. Here, each fraud proof invalidates an entire transaction block (i.e., hundreds of operations) to simulate large scale failures. We further evaluate performance with a single shard, so that shard commander failures have the highest impact.

Figure 7 shows that the performance degradation is negligible during failures, as in a real-world setting the fraud rate would most likely not exceed one failure a second. The intuition behind this result is that discarded operations do not necessarily cause an abort, but extend the transactions challenge period and allow the new shard commander to resubmit the operation. Because the performance is mostly dominated by verifying transactions, shard failures come with a rather small penalty on the throughput generated by the re-issuing of operations and the increased key block size.

6 Related Work

6.1 Byzantine Fault-Tolerant State Machine Replication

Practical Byzantine Fault-Tolerance (PBFT) [5] was one of the first Byzantine fault-tolerant consensus protocols and is stil widely used today. Several improvements to PBFT have been proposed over the last years. For example, Zyzzyva [22] avoids the third round of messages in the absence of failures using speculative execution. Aardvark [6] adds additional robustness by making clients digitally sign their requests and frequently rotating leadership. HotStuff [41] reduces message complexity during leader election among other modifications to the protocol.

6.2 Sharding Database Systems

The concepts of sharding and distributed transactions have been widely studied with regard to conventional database systems. Sharding was first popularized by systems like Chord [36] and Mercury [3]. Later work introduced systems, such as Chubby [4], that provide serializable transactions on top of sharded systems. More recent work aims to improve the performance by reducing coordination [9, 29] or relying on loosely synchronized clocks [8].

6.3 Sharding Blockchains

Several other sharding solutions have been proposed for permissionless and permissioned blockchain systems.

Monoxide [40] is an approach that has some similarities with BitWeave. Monoxide breaks up the workload across independent "consensus zone", each having its own set of miners. Unlike BitWeave, Monoxide does not support generalized transactions, but only money transfers between exactly two accounts. For a cross-zone transaction, the transactions are first processed in the source account's zone and then forwarded to the target account's zone together with a Merkle proof of the transaction's inclusion. At some point, the transaction will be included in the source and the target zone, however, the protocol does not provide an upper time-bound for this. In BitWeave, the confirmation period provides a time-bound for when a transaction is expected to be confirmed. Furthermore, the transaction processing scheme proposed in Monoxide is susceptible to recursive invalidation of dependent transactions in the case of zone-forks. In BitWeave, keyblocks create consistent cuts which, in combination with the validation delays, prevent such issues. Another challenge with Monoxide's design is that its independent zones naturally partition the mining power of the blockchain system, which dilutes the overall security of the system. The authors address this by assuming the majority of miners will work on all zones at the same time, which requires miners to possess large amounts of processing power for verification to maintain the same security guarantees as Bitcoin. This encourages mining centralization for high throughput, giving up the key property of blockchains.

Elastico [26], OmniLedger [21], and Chainspace [1] are in a similar class of scalability solutions that propose dividing the nodes in a system into small committees, each of which performs a Byzantine consensus protocol for intra-shard consensus. The Elastico protocol, the first of such solutions, proceeds in the following fashion: protection against Sybils is achieved using an "identity chain" based on Proof-of-Work. It then pseudo-randomly assigns nodes to committees that perform PBFT in rounds until all the nodes in the system agree on a final change set to be committed. The protocol then re-assigns committees and restarts the process for the next set of transactions. Chainspace assumes a permissioned system and does not discuss shard committee selection.

OmniLedger makes further improvements on top of Elastico, such as using RandHound to better seed for randomness in shard assignments and helps ameliorate some security compromises introduced by Elastico's small committee sizes. However, OmniLedger still adds several layers of complexity to public blockchains. This complexity is especially salient when examining the need for OmniLedger to have day-long epochs because of the amount of overhead required for bootstrapping at the beginning of an epoch, which makes it susceptible to quick-responding attackers.

Zilliqa [37] shards transactions, but not state. They rely on a similar mechanism as Omniledger to assign nodes to shard, but on a different cross-shard commit protocol. Instead of splitting the state of the system across shards, they only split the transaction workload and replicate state among all nodes. Each shard then processes a subset of all transaction for a specific epoch, and merges their resulting state with other shards at certain checkpoints. At a high level, the protocol allows a particular shard to lock parts of the state to prevent concurrent modification of the same data entries. Zilliqa relies on a dataflow-based programming model to implement this scheme efficiently.

6.4 Sidechains and Off-Chain Mechanisms

Yet another category of scalability solutions for permissionless chains is that of federated chains, or side-chains, which solutions layered on top of existing blockchain systems. In general, these solutions lock funds on some existing system and facilitate the fast transactions between parties through an off-chain protocol. Only the amount locked on the base chain is allowed to be exchanged in these systems, and a tally of balances is kept for when it is time to settle. On settling, the amount apportioned to the settler as denoted by her balance in the sidechain is unlocked on the main chain and returned to the settler. Plasma [34] is one notable example of a side-chain that can be anchored onto Ethereum. Payment- and state-channels [35, 25, 28] build networks of peer-to-peer relationships to process certain operations without the involvement of the main blockchain.

These approaches are similar to BitWeave at first sight, but differ significantly in functionality and safety. First, BitWeave is a holistic protocol that ensures the main chain is always able to process fraud proofs. Most side channel protocols, on the other hand, assume the main chain will always be able to process fraud proofs, which might not be true during high contention. Second, BitWeave allows natively supports smart contracts. While state channels do support smart contracts, they are currently very limited in programmability and require knowing all participants of an off-chain contract at setup.

6.5 Audit Mechanisms

Fraud proofs were first introduced in work auditing centralized services [42, 24, 27], such as a filesystem. Here, the centralized system provides an auditable log of modifications to its clients. Clients communicate through channels hidden from the centralized service to exchange the log data they receive from the service and detect discrepancies in the log. Unlike blockchains, this line of work does not provide mechanisms to recover from Byzantine failures but just means to detect them. The underlying assumption is that rational operators of service will not perform fraudulent behavior knowing they will be caught. Recent work, such as BlockchainDB [11], Arbitrum [17], and FalconDB [32], has extended this scheme to include replication and failure recovery coordinated by a global blockchain.

7 Discussion and Future Work

7.1 Shortening CHALLENGE Stages

Recent work investigated how to reduce confirmation times in Bitcoin. ByzCoin [20] reduces confirmation times by establishing a set of validating nodes. Members of validator committee are selected by observing which entities mined the last n blocks on some *identity chain* based on PoW. These validators then issue blocks using a conventional quorum based consensus protocol. Thunderella [31] makes this scheme more resilient against churn, i.e., validators leaving and rejoining the network. The ByzCoin approach is directly applicable to BitWeave. Here a committee generates key blocks that are guaranteed to be benign, which reduces validation time from 6 epochs to just one.

Another way to reduce CHALLENGE periods significantly is to tighten to bound on network propagation delays through more efficient relay networks. Currently, BitWeave assumes a network propagation delay of about three minutes. BloXroute [19] is one approach to reduce delays through a centralized but untrusted relay network. Similar to solutions layered on top of a blockchain, these "layer 0" solutions are orthogonal to BitWeave.

7.2 Reducing Chain Size

A common problem with blockchains is that they continuously grow in size, which results in a large storage overhead for nodes participating in the protocol. This problem is exacerbated in high-throughput blockchains, such as BitWeave, as significantly more transactions are admitted to the chain.

BitWeave can enable nodes to reduce storage size significantly through on-chain snapshots. Snapshots allow nodes to track a lightweight representation of the blockchain's stable prefix. Shard commanders regularly publish a snapshot that encapsulates the current state of the shard. Once the snapshot has been validated and is buried deep enough in the chain to be considered stable, nodes discard all microblocks of that shard up until the snapshot. They merely maintain a representation of the main chain that verifies that the snapshot has been generated by an authorized source and no fraud proofs have been raised against it. Assuming the number of accounts stays the same, this stops the blockchain from linearly growing with the number of transaction to some constant size.

7.3 Adapting to Changing Workloads

The workload of a blockchain varies, which can affect BitWeave's performance and safety. For example, when a token goes on sale many clients may issue transactions for a specific smart contract. This is a problem because one shard commander might not be able to process all the incoming requests. On the other hand, if overall the number of transactions and thus active participants in the network declines, there may not be a sufficient number of validators per shards to ensure safety.

The BitWeave protocol can be extended to support a varying number of shards to address this. As in conventional database systems, we differentiate between virtual shards, which are of a constant number, and logical shards, that contain one or more virtual shards and are assigned to logical shards [36]. Logical shards are then assigned to a single validator like before. Nodes that get assigned to a different shard leverage snapshots so that they do not have to parse the entire shard's chain.

We propose a load balancer function as part of the BitWeave protocol that follows a similar mechanism as the difficulty adjustment in Bitcoin. While the difficulty adjustment in Bitcoin is a function of epoch length, the load balancer in BitWeave is a function of the number of transactions in and length of the last epoch. The function deterministically defines how many logical shards are supposed to exist and how the virtual shards are assigned to them. A lower number of logical shards reduces the amount of concurrency in the system while allowing for easier validation of shard commanders.

8 Conclusion

This paper introduced BitWeave, a blockchain protocol that can scale linearly with the number of shards while maintaining Byzantine fault-tolerance. BitWeave's core design goal is to allow small entities to participate in the consensus mechanism, arguably the key property of public blockchains. The protocol further provides sound incentive mechanisms and the same security as Bitcoin. Unlike previous solutions, the protocol does not dilute mining power and supports a fully decentralized network. Our experimental evaluation of BitWeave shows that it can support realistic workloads and is flexible enough to be applicable to a wide variety of blockchains, including Bitcoin and Ethereum.

References

[1] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform. *arXiv preprint arXiv:1708.03778,* 2017.

[2] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. Fraud Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities. *CoRR,* abs/1809.09044, 2018.

[3] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: supporting scalable multi-attribute range queries. *SIGCOMM Conference,* pages 353-366, Portland, Oregon, September 2004.

[4] Michael Burrows. The Chubby Lock Service for Loosely-Coupled Distributed Systems. *Symposium on Operating System Design and Implementation,* pages 335-350, Seattle, Washington, November 2006.

[5] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. *Symposium on Operating System Design and Implementation,* pages 173-186, New Orleans, Louisiana, February 1999.

[6] Allen Clement, Edmund L. Wong, Lorenzo Alvisi, Michael Dahlin, and Mirco Marchetti. Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults. *Symposium on Networked System Design and Implementation,* pages 153-168, Boston, Massachusetts, April 2009.

[7] Matt Corallo. The Fast Internet Bitcoin Relay Engine (FIBRE). http://www.bitcoinfibre.org/.

[8] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson C. Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems,* 31(3):8, 2013.

[9] James Cowling and Barbara Liskov. Granola: low-overhead distributed transaction coordination. *USENIX Annual Technical Conference,* 2012.

[10] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed E. Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On Scaling Decentralized Blockchains - (A Position Paper). *Financial Cryptography and Data Security,* pages 106-125, Christ Church, Barbados, February 2016.

[11] Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy. BlockchainDB - A Shared Database on Blockchains. *Proceedings of the VLDB Endowment,* 12(11):1597-1609, 2019.

[12] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. Bitcoin-NG: A Scalable Blockchain Protocol. *Symposium on Networked System Design and Implementation,* pages 45-59, Santa Clara, California, March 2016.

[13] Ittay Eyal and Emin Gün Sirer. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. *Financial Cryptography and Data Security,* pages 436-454, Christ Church, Barbados, March 2014.

[14] Ittay Eyal and Emin Gün Sirer. Majority is not enough: bitcoin mining is vulnerable. *Communications of the ACM,* 61(7):95-102, 2018.

[15] The Ethereum Foundation. ERC20 Token Standard. https://theethereum.wiki/w/index.php/ERC20_Token_Standard, 2018.

[16] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. Decentralization in Bitcoin and Ethereum Networks. *Financial Cryptography and Data Security,* pages 439-457, Porta Blancu, Curaçao, February 2018.

[17] Harry A. Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. Arbitrum: Scalable, private smart contracts. *USENIX Security Symposium,* pages 1353-1370, Baltimore, Maryland, August 2018.

[18] David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. *STOC,* pages 654–663, 1997.

[19] Uri Klarman, Soumya Basu, Aleksandar Kuzmanovic, and Emin Gün Sirer. bloXroute: A Scalable Trustless Blockchain Distribution Network. *bloXroute white paper,* 2018.

[20] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. *USENIX Security Symposium,* pages 279-296, Austin, Texas, August 2016.

[21] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. *IEEE Symposium on Security and Privacy,* pages 583-598, San Francisco, California, May 2018.

[22] Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund L. Wong. Zyzzyva: speculative byzantine fault tolerance. *Symposium on Operating Systems Principles,* pages 45-58, Stevenson, Washington, October 2007.

[23] Nir Kshetri. Blockchain's roles in meeting key supply chain management objectives. *International Journal of Information Management,* 39:80–89, 2018.

[24] Jinyuan Li, Maxwell N. Krohn, David Mazières, and Dennis E. Shasha. Secure Untrusted Data Repository (SUNDR). *Symposium on Operating System Design and Implementation,* pages 121-136, San Francisco, California, December 2004.

[25] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter R. Pietzuch. Teechain: a secure payment network with asynchronous blockchain access. *Symposium on Operating Systems Principles,* pages 63-79, Huntsville, Ontario, Canada, October 2019.

[26] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A Secure Sharding Protocol For Open Blockchains. *Computer and Communications Security,* pages 17-30, Vienna, Austria, October 2016.

[27] David Mazières and Dennis E. Shasha. Building secure file systems out of Byzantine storage. *ACM Symposium on Principles of Distributed Computing,* pages 108-117, Monterey, California, July 2002.

[28] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. Sprites: Payment Channels that Go Faster than Lightning. *CoRR,* abs/1702.05812, 2017.

[29] Shuai Mu, Yang Cui, Yang Zhang, Wyatt Lloyd, and Jinyang Li. Extracting More Concurrency from Distributed Transactions. *Symposium on Operating System Design and Implementation,* pages 479-494, Broomfield, Colorado, October 2014.

[30] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[31] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. *Annual International Conference on the Theory and Applications of Cryptographic Techniques,* pages 3–33, 2018.

[32] Yanqing Peng, Min Du, Feifei Li, Raymond Cheng, and Dawn Song. FalconDB: Blockchain-based Collaborative Database. *SIGMOD International Conference on Management of Data,* pages 637-652, Portland, Oregon, June 2020.

[33] Soujanya Ponnapalli, Aashaka Shah, Amy Tai, Souvik Banerjee, Vijay Chidambaram, Dahlia Malkhi, and Michael Wei. Scalable and Efficient Data Authentication for Decentralized Systems. *arXiv preprint arXiv:1909.11590,* 2019.

[34] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper,* 2017.

[35] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf, 2016.

[36] Ion Stoica, Robert Tappan Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Conference,* pages 149-160, San Diego, California, August 2001.

[37] Zilliqa Team. The Zilliq Technical Whitepaper. *Technical Report,* 2017.

[38] Jason Teutsch, Vitalik Buterin, and Christopher Brown. Interactive coin offerings. *arXiv preprint arXiv:1908.04295,* 2019.

[39] Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. SoK: Sharding on Blockchain. *Advances in Financial Technologies,* pages 41-61, Zürich, Switzerland, October 2019.

[40] Jiaping Wang and Hao Wang. Monoxide: Scale out Blockchains with Asynchronous Consensus Zones. *Symposium on Networked System Design and Implementation,* pages 95-112, Boston, Massachusetts, February 2019.

[41] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. HotStuff: BFT Consensus with Linearity and Responsiveness. *ACM Symposium on Principles of Distributed Computing,* pages 347-356, Toronto, Canada, July 2019.

[42] Aydan R. Yumerefendi and Jeffrey S. Chase. Strong Accountability for Network Storage. *Conference on File and Storage Technologies,* pages 77-92, San Jose, California, February 2007.