Internet of Services: The Next-generation, Secure, Highly Scalable Ecosystem for Online Services

The Internet of Services Foundation

December 31, 2017

Abstract

Despite the recent hype in the cryptocurrency market, the underlying blockchain technology is still at an early stage and is far from mass adoption. One of the well-recognized issues with current blockchain technologies is scalability. Without the capability to process large volumes of transactions swiftly, heavy usage services like Facebook, Amazon, and digital asset exchanges are nearly impossible to deploy onto the blockchain.

In this paper, we propose the Internet of Services ("**IOS**"), an innovative and secure blockchain paradigm designed to provide horizontal scalability and high transaction throughput. By implementing our novel sharding architecture and consensus mechanism, the IOS system is able to process up to 100,000 secure transactions per second.

This work makes the following contributions. It introduces:

1. *Efficient Distributed Sharding* (EDS) - an innovative sharding scheme that makes shards sufficiently large and strongly bias-resistant via a combination of a client-server randomness scavenging mechanism and leader election via cryptographic sortition.
2. *TransEpoch* - a secure validators-to-shards assignment during epoch transitions while maintaining transaction operability.
3. *Atomix* - an improved two-step inter-shards atomic commit protocol that guarantees transaction atomicity across shards in Byzantine settings.
4. *Micro State Block* (MSB) - a novel mechanism to minimize the storage and bootstrapping costs for validators.
5. *Proof-of-Believability* (PoB) - a groundbreaking Byzantine consensus protocol with a Believable-First approach that guarantees safety and liveness of the system while largely maximizes the transaction throughput by size-one-shard.

Note: the IOS is a work in progress. Active research is under way, and new versions of this paper will be updated at http://iost.io. For comments and suggestions, contact us at team@iost.io.

Contents

1 Background

Excessive commission fees, privacy violations, frauds and censorship are common issues encountered during daily interactions with centralized online service providers. Given these well-recognized problems with centralization, a wide range of blockchain technologies attempting to resolve these issues have been developed since the launch of Bitcoin [14] in 2008. Specialized projects like SteemIt [26], Bitshares [27], and Syscoin [20], as well as more versatile projects like Ethereum [25] and EOS [10], are some of many examples. However, most of those attempts are either too specialized in certain applications, or burdened by low transaction throughput. Due to these limitations in flexibility and transaction throughput, it is impossible for developers and enterprises to bring heavy services like Facebook or Amazon onto the blockchain - not to mention something more complicated like digital asset exchanges.

The heart of the scalability issue lies in the fundamental design of these existing blockchain technologies -- their consensus protocols and blockchain architectures. Most of the existing blockchain technologies face two major challenges in their way of scaling up: a) every full node must store the entire ledger in order to participate; b) every participating node in the network is obligated to handle every transaction. Since all the participating nodes are essentially conducting the same work, the number of transactions the system can process will not exceed that of a single node. Moreover, the growing size of the blockchain increases the requirements and costs of storage space, bandwidth, and computational resources for a node to fully participate in the network. The increasing mining cost will inevitably make the participation in the network become a privilege for the few, leading us straight back to the problem of centralization.

The IOS is designed to fill the void. In our vision, the IOS is a next-generation blockchain technology that provides the network infrastructure to support a service-oriented ecosystem. The IOS platform not only provides its users a completely decentralized way to exchange online services and digital goods, but also enables developers to deploy large scale dApps with the ability to support massive number of users. With a series of groundbreaking innovations, such as Efficient Distributed Sharding (“EDS”) and Believable-First consensus approach, we are able to increase the system’s throughput enormously while guaranteeing security.

We developed EDS based on the sharding technique. It is widely adopted in distributed systems and databases to enable parallel transaction processing. Inspired by the classic

“Divide and Conquer” principle in computer science, sharding is a technique that partitions the entire IOS network into certain numbers of subspaces called *shards*. We can consider each shard as a miniature network that runs its own consensus protocol in parallel. Instead of having the entire network validating the same set of transactions, subsets of transactions can be handled by various consensus groups simultaneously. Therefore, the throughput of the system can be significantly enhanced even if the size of the network and number of transactions grow rapidly. Moreover, in order to ensure the network is divided homogeneously, we have developed a Bias-Resistant Distributed Randomness protocol in order to introduce unbiased and transparent randomness into the sharding process.

With the EDS, the IOS is also packed in a very powerful arsenal of technologies that empowers deployment of large scale dApps with high-performance and flexibility. It allows developers to build a wide range of products: from counterparts of traditional monopolistic online service providers to brand new business models that would have been considered impossible previously. Admittedly, running such services could be quite expensive when the size of the IOS network is relatively small. However, with increasing number of nodes and resources in the network, the cost of running such large scale dApps might be greatly reduced. Additionally, there are many benefits including: avoiding cyber attacks, high-level of data security, and the immutable property.

During the development of the IOS blockchain, we thoroughly examined all currently available solutions in order to learn from previous attempts.

2 Related Work

2.1 State Machine Replication

In a nutshell, blockchain technologies are a branch of state machine replication protocols. Every state machine replication protocol has to satisfy two important properties:

1. *Safety*, i.e., all servers in the network have the same record of transactions;
2. *Liveness*, i.e., whenever a client submits a transaction, the transaction will be documented into the log quickly.

There are two fundamentally different ways to achieve state machine replication: classical-style consensus and blockchain-style consensus. Classical-style consensus generally applies Paxos-like algorithms and is used in the permissioned setting where there is a priori knowledge of the consensus nodes known by the system. An example of this application would be the servers at software companies like Amazon, where their servers collectively use a classical way to replicate and store information, and the classical algorithm establishes the fundamentals to form a consensus of the ordering of their data.

2.2 Bitcoin and Proof-of-Work

Satoshi Nakamoto was the first to introduce Bitcoin as a solution to establish consensus in the permissionless setting , e.g. any node can freely join and leave the network without a priori knowledge of the consensus nodes. The network underlying Bitcoin, *blockchain* improves the scale of distributed systems without human involvement by providing economic incentives to the servers, dubbed miners in Bitcoin's settings. Miners in the Bitcoin network form consensus by calculating partial hash collisions with a certain difficulty level. The chain with the greatest cumulative difficulty would be acknowledged by other nodes as the consensus result. This solution is named Proof-of-Work (PoW), which in essence is to have all the nodes in the network contribute their computing power as a way to earn incentives and thus determine the ordering of transactions for the whole system. A benefit of PoW is its ability to defend against Sybil attack in a permissionless setting [14]. Despite its advancements in scale and security, Bitcoin has a few major drawbacks: (1) unlike other modern cryptocurrencies, it takes more than an hour to confirm a transaction according its configuration; (2) it is difficult to develop various applications upon Bitcoin network; (3) the consensus mechanism wastes too much energy, i.e., it costs more than two million dollars per day in electricity. More importantly, earlier works show that

Bitcoin-style blockchain must have a sufficiently large interval to retain security [15] [16].

Therefore, Bitcoin would not be a good replacement for the current centralized system to support day-to-day applications and large transaction volume.

2.3 Proof-of-Stake

The concept of Proof-of-Stake was first discussed on an online blockchain forum [28] and was adopted by a few cryptocurrencies like PPcoin [21], PeerCoins[22] and Nxt [5]. The idea of PoS is essentially one vote per unit of stake, such that for each validator, owning more stake will have higher voting power. Therefore, validators have no economic incentive to harm the whole blockchain network. For attackers, the cost of attack is huge because they have to own the majority of the stakes. In the early development, Proof-of-Stake consensus mechanism is known for being vulnerable to “nothing-at-stake” attacks, where servers are able to vote on multiple blocks at the same time with no incentive to converge, damaging the security of the blockchain. Later work solves the problem using *slasher* [3], which enforces a punishment for violating nodes. Many other work are also described as the ad hoc application of Proof-of-Stake [1–3,24][11][9]. Although PoS fulfills the *liveness* of the replicated state machine protocol, it still faces challenges like centralization and security problems. For instance, validators possessing more tokens will be more likely to generate new blocks and get richer, leading to a potential centralization problem. Furthermore, previous work shows that Proof-of-Stake protocol can only be a provably secure and robustly configured consensus protocol if its token is not being exchanged too frequently [17] , which potentially implies that there is a ceiling throughput for Proof-of-Stake in order to preserve security.

3 Blockchain Architecture

In summary, our blockchain architecture, namely IOSChain, contains the following contributions/designs:

- In Section 4 and 5, We present *EDS* - a novel scheme to form shards (subsets of validators to record state and process transactions) that are both sufficiently large and strongly bias-resistant using a combination of a client-server randomness scavenging mechanism enabling client to gather randomness on demand (DRP subprotocol), and a VRF-based leader election via cryptographic sortition.
- In Section 6, We present *TransEpoch* - a secure validators-to-shards assignment during epoch transitions while maintaining system operability.
- In Section 7, We present *Atomix* - a novel two-step inter-shard atomic commit protocol that guarantees transaction atomicity in Byzantine setting.
- In Section 8, We present *Proof-of-Believability* - a novel Byzantine consensus protocol with a Believable-First approach that guarantees safety and *liveness* of the system while largely maximizing the transaction throughput by size-one-shard.
- In Section 9, We present *MSB* - a novel mechanism to minimize the storage and bootstrapping costs for validators.

4 Efficient Distributed Sharding

To generate a seed for distributed sharding without relying on a Proof-of-Work mechanism [12] or a trusted randomness beacon [6], we use a distributed randomness generation protocol that is performed by validators collectively.

Requirement We require that the distributed randomness generation protocol is

- Scalable
- Unpredictable
- Unbiased
- Third-party verifiable

We proposed a novel Distributed Randomness Protocol (DRP) that addresses each requirement above.

Since DRP relies on a leader to run the protocol, we need an appropriate mechanism to elect one of the validators to be the leader. If we use a deterministic method for leader election, then the malicious nodes can enforce up to f out of n failures in the worst case by refusing to run the protocol, resulting in up to $1/3\ n$ failures given our threat model. Thus, the selection mechanism itself must be unbiased and unpredictable. To conquer this predicament, we use cryptographic sortition [8] to elect the leader.

4.1 Algorithm - Leader Election Protocol

Algorithm 1: **Leader Election Protocol**

Inputs:

1) $conf_e$ is the configuration containing all properly registered validators
2) v is a view counter
3) i is a validator
4) sk_i is the private key for i
5) e is the current epoch
6) Δ is the synchrony bound

Output: a validator v who has the minimum-value valid lottery to run DRP

1) For each e, each i computes a lottery $lottery_{i,e,v} = VRF_{sk_i}(conf_e \parallel v)$

2) Then for a time Δ, the validators gossip these lotteries with each other.
3) After Δ, they lock in the minimum-value valid lottery they have seen so far
4) Validators accept the validator who has that lottery as the leader of the DRP protocol execution.

5) If the elected validator fails to start DRP within another Δ, validators consider the current run as failed and ignore this validator for the rest of the epoch e. Then the nodes increment the view number to $v + 1$ and re-run the leader election lottery.

In the end, after the validators have successfully completed a run of DRP and the leader has broadcasted rnd_e together with its correctness proof, each of the n properly registered validators can first verify and then use rnd_e to compute a permutation π_e of $1, \ldots, n$ and subdivide the result into m equally-sized buckets, therefore determining its mapping of nodes to shards. We make the proof to argue the security of the approach.

4.2 Analysis

Security Theorem 1: *leader election mechanism gives unpredictability, unbiasability, and third-party verifiability.*

Proof: Each validator can produce only a single valid lottery per view v in a given epoch e. A single valid ticket will be produced because the VRF-based leader election only starts after the valid identities have been fixed in the identity blockchain. Moreover, as the output of a VRF is unpredictable as long as the private key sk_i is kept secret, the tickets of non-colluding nodes, hence the outcome of the lottery are also unpredictable. The synchrony bound Δ guarantees that the ticket of an honest leader is seen by all other honest validators. If the malicious wins the lottery, then it can either decide to cooperate and run the DRP protocol or decide to fail it which would exclude that particular node from participating for the rest of the epoch. ■

After a successful run of DRP, the adversary learns the randomness and the sharding mapping, but the benefit to the adversary is minimal. The adversary can either decide to comply and broadcast the random value or it can withhold it in the hope of winning the lottery again and obtaining a sharding mapping that fits its plan better. However, the probability that an adversary wins the lottery a times consecutively is upper bounded by the exponentially decreasing term $(f/n)^a$. Therefore, after only a few rounds of the lottery, an honest node wins w.h.p and coordinates the sharding. Finally, we remark that an adversary cannot collect random values from multiple runs and then choose the one it favors because validators need to accept only the latest random value that matches their

view number v.

5 Distributed Randomness Protocol

We then formally describe Distributed Randomness Protocol (DRP). DRP consists of two phases - randomness generation and randomness verification. DRP use a client-server model, in which a client invokes the services of a group of DRP servers to produce a random value.

5.1 Overview

Overview of the protocol run is described below:

1) (server) Each server chooses its random input and generates shares only for other group members using *Publicly Verifiable Secret Sharing Scheme [19]*. The encrypted shares is sending to the together with the *NIZK* [23] proofs.
2) (client) The client chooses a subset of server inputs from each group, excluding servers that does not respond on time or with proper values, thus fixing each group's secret and consequently the output of the protocol.
3) (server) After the client receives a sign-off on its choice of inputs in a global run of CoSi, the servers decrypt and send their shares to the client.
4) (client) The client then combines the recovered group secrets to produce the final random output Z. The client documents the run of the protocol in a transcript L by recording the messages it sends and receives. The transcript serves as a third party verifiable proof of the produced randomness.

We present Randomness Generation first. DRP-generation consists of three inquiry-response stages between the client and the servers and client's randomness recovery; The communications are signed by the sender, messages from the client to servers contain the session identifier, and messages from servers to the client contain a response identifier which is the hash of the previous client message. We implicitly assume that client and servers always verify message signatures and session and response identifiers and that they mark non-authentic or replayed messages and ignore them from the rest of the protocol run.

5.2 Algorithm - Distributed Randomness Generation and Verification

Algorithm 2: Distributed Randomness Generation

Inputs:

1) G is a group of large prime order q with generator G
2) denote the list of nodes
3) $S = N \setminus \{0\}$ is the list of servers

4) f is the maximum number of permitted Byzantine nodes. We require that $n = 3f + 1$
5) (x_0, X_0) is the key pair of the client
6) (x_i, X_i) is the one of server $i > 0$
7) $T_l \subset S$, with $l \in \{0, \ldots, m-1\}$, be pairwise disjoint trustee groups
8) $t_l = |T_l|/3 + 1$ be the secret sharing threshold for group T_l
9) $X = (X_0, \ldots, X_{n-1})$ is the list of public keys
10) $T = (T_0, \ldots, T_{m-1})$ is the server grouping
11) u is a purpose string
12) w is a timestamp
13) $C = (X, T, f, u, w)$ denotes the publicly available session configuration
14) $H(C)$ is the session identifier.
15) Note that the session configuration and consequently the session identifier have to be unique for each protocol run. We assume that all nodes know the list of public keys X.

Outputs: random string Z which is publicly verifiable through a transcript L

1) **Initialize**. For the client:
   a) Set the values in C and choose a random integer $r_T \in_R \mathbb{Z}_q$ as a seed to pseudorandomly create a balanced grouping T of S.
   b) Mark C in L.
   c) Prepare the message $\langle l_1 \rangle_{x_0} = \langle H(C), T, u, w \rangle_{x_0}$
   d) record $\langle l_1 \rangle_{x_0}$ in L, and broadcast to all servers.
2) **Share**. for each trustee $i \in T_l$
   a) Map $H(C)$ to a group element $H \in G^*$, set $t_l = |T_l|/3 + 1$, and (randomly) choose a degree $t_l - 1$ secret sharing polynomial . The secret to-beshared is $S_{io} = G^{s_i(0)}$.
   b) Create polynomial commitments A_{ik}, for all $k \in \{0, \ldots, t_l - 1\}$, and compute encrypted shares $\widehat{S_{ij}} = X_j^{s_i(j)}$ and consistency proofs $\widehat{P_{ij}}$ for all $j \in T_l$.
   c) Choose $v_i \in_R \mathbb{Z}_q$ and compute $V_i = G^{v_i}$ as a Schnorr commitment.
   d) Prepare the message $\langle R_{1i} \rangle_{x_i} = \langle H(1), (\widehat{S_{ij}}, \widehat{P_{ij}})_{j \in T_l}, (A_{ik})_{k \in \{0, \ldots, t_l - 1\}}, V_i \rangle_{x_i}$ and send it back to the client.
3) **Commit**. Client commits to the set of shared secrets and asks servers to co-sign:
   a) Record each received $\langle R_{1i} \rangle_{x_i}$ message in L.

b) Verify all $\widehat{S_{ij}}$ against $\widehat{P_{ij}}$ using X_i and A_{ik}. Buffer each $H^{s_i(j)}$ created in the process. Mark each share not verified as invalid, and do not forward the corresponding tuple $(\widehat{S_{ij}}, \widehat{P_{ij}}, H^{s_i(j)})$ to the respective trustee.

c) Create the commitment to the final list of secrets $T' = (T'_0, ..., T'_{m-1})$ by randomly selecting $T'_l \subset T_l$ such that $|T'_l| = t_l$ for all $l \in \{0, \ldots, m-1\}$.

d) Compute the aggregate Schnorr commit $V = \Pi_i V_i$ and the Schnorr challenge $c = H(V \parallel H(C) \parallel T')$.

e) Prepare the message $\langle l_{2i} \rangle_{x_0} = \langle H(C), c, T', (\widehat{S_{ji}}, \widehat{P_{ji}}, H^{s_j(i)})_{j \in T'_l} \rangle_{x_0}$, record it in L, and send it to trustee $i \in T_l$.

4) **Acknowledge**. Each trustee $i \in T_l$ ack the client's commitment:

a) Check that $T'_l = t_l$ for each T'_l in T' and that $f + 1 \leq \sum_{l=0}^{m-1} t_l$. Abort if any of those conditions does not hold.

b) Compute the Schnorr response $r_i = v_i - cx_i$.

c) Prepare the message $\langle R_{2i} \rangle_{x_i} = \langle H(l_{2i}), r_i \rangle_{x_i}$ and send it back to the client.

5) **Decrypt**. Client requests the decryption of the secrets from the trustees by showing a valid Schnorr signature:

a) Record each received $\langle R_{2i} \rangle_{x_i}$ message in L.

b) Compute the aggregate Schnorr response $r = \sum_i r_i$ and create a list of exceptions E that contains information on missing server commits and/or responses.

c) Prepare the message $\langle l_3 \rangle_{x_i} = \langle H(l_{2i}), r_i \rangle_{x_i}$,record it in L, and broadcast it to all servers.

6) **Share Decrypt**: To decrypt shares, for each trustee $i \in T_l$

a) Check that (c, r) forms a valid Schnorr signature on T' taking exceptions recorded in E into account and verify that at least $2f + 1$ servers signed. Abort if any of those conditions does not hold.

b) Check for all $j \in T'_l$ that $\widehat{S_{ji}}$ verifies against $\widehat{P_{ji}}$ using $H^{s_j(i)}$ and public key X_i.

c) If the verification fails, mark $\widehat{S_{ji}}$ as invalid and do not decrypt it. Otherwise, decrypt $\widehat{S_{ji}}$ by computing $S_{ji} = (\widehat{S_{ji}})^{x_i^{-1}} = G^{s_j(i)}$ and create a decryption consistency proof P_{ji}.

d) Prepare the message $\langle R_{3i} \rangle_{x_i} = \langle H(l_3), (S_{ji}, P_{ji})_{j \in T'_l} \rangle_{x_i}$ and send it back to the client.

7) **Recovery**. To construct collective randomness, for the client :

a) Record all received $\langle R_{3i} \rangle_{x_i}$ messages in L.

b) Check each share $S_{::}$ against $P_{::}$ and mark invalid ones.

c) Use Lagrange interpolation to recover the individual S_{i0} that have enough valid shares S_{ij} and abort if one of the secrets previously committed to in T' cannot be reconstructed.

d) Compute the collective random value as $Z = \prod_{i \in \cup T'_l} S_{i0}$, and publish Z and L

Then we present the Randomness Verification Protocol.

Algorithm 3: **Distributed Randomness Verification**

Inputs: Output of the distributed randomness generation protocol

1) collective randomness Z
2) Transcript $L = (C, \langle l_1 \rangle_{x_0}, \langle R_{1i} \rangle_{x_i}, \langle l_{2i} \rangle_{x_0}, \langle R_{2i} \rangle_{x_i}, \langle l_3 \rangle_{x_0}, \langle R_{2i} \rangle_{x_i})$

Outputs: Validity of the collective randomness Z

1) Verify the values of arguments included in the session configuration $C = (X, T, f, u, w)$. Specifically, check that $|X| = n = 3f + 1$, that groups T_l defined in T are non-overlapping and balanced, that $|X| = \sum_{l=0}^{m-1} |T_l|$, that each group threshold satisfies $t_l = |T_l|/3 + 1$, that u and w match the intended use of Z, and that the hash of C matches $H(C)$ as recorded in the messages.
2) Verify all signatures of $\langle l_1 \rangle_{x_0}, \langle R_{1i} \rangle_{x_i}, \langle l_{2i} \rangle_{x_0}, \langle R_{2i} \rangle_{x_i}, \langle l_3 \rangle_{x_0}, \langle R_{2i} \rangle_{x_i}$. Ignore invalid messages for the rest of the verification.
3) Verify that $H(l_1)$ matches the hash recorded in R_{1i}. Repeat for L_{1i} and R_{1i}, and L_3 and R_{3i}. Ignore messages that do not include the correct hash.
4) Check that T' contains at least $f + 1$ secrets, that the collective signature on c is valid and that at least $2f + 1$ servers contributed to the signature (taking into account the exceptions in E). 5) Verify each recorded encrypted share $\widehat{S_{ij}}$, whose secret was chosen in T', against the proof P' using X_i and A_{ik}. Abort if there are not enough shares for any secret chosen in T'.
5) Verify each recorded decrypted share S_{ij} against the proof P_{ij} where the corresponding $\widehat{S_{ij}}$ was found to be valid. Abort if there are not enough shares for any secret chosen in T'.
6) Verify Z by recovering Z' from the recovered individual secrets S_{i0} and by checking that $Z = Z'$. If the values are equal, then the collective randomness Z is valid. Otherwise, reject Z

5.3 Security Properties

DRP provides the following security properties:

- Availability. For an honest client, the protocol successfully finishes and produces the final random output Z w.h.p..
- Unpredictability. No entity learns anything about the final random output Z, except with negligible probability, until the secret shares are revealed.

- Unbiasability. The final random output Z represents an unbiased, uniformly random value, except with negligible probability.
- Verifiability. The collective randomness Z is third-party verifiable against the transcript L, that serves as an unforgeable confirmation that the documented set of participants ran the protocol to produce the one-and-only random output Z, except with negligible probability.

6 Operability During Epoch Transitions

Our shards are reconfigured dynamically - the assignments of validators to shards changes in each epoch e, which results in an idle period during which the network cannot process any transactions until enough validators have finished reconfiguration and bootstrapping.

To maintain operability during idle phases, we use a method to select a subset of the committee to be swapped out and replaced with new members per epoch. This enables the remaining validators to continue offering service while the newly joined nodes are bootstrapping. In order to achieve this continuous operation we need to swap out at most $\frac{1}{3}$ of the shard's size ($\approx \frac{n}{m}$).

Batch Size Risk batch size is highly relevant to the safety of the system.

- When the swap batch size grows, the risk increases as the number of remaining honest validators will not be sufficient to reach consensus
- When the swap batch size grows, the downloading and bootstrapping information will cause network stress increases

To minimize the chances of a transitory loss of liveness of the system, we used a novel shard assignment algorithm for validators - TransEpoch.

5.4 Algorithm - TransEpoch

Algorithm 4: **TransEpoch**

Inputs:

1) n is the number of nodes
2) m is the size of shards
3) k is the [illegible] validators that will be [illegible]

3) k is the the swap-out batch size, i.e., the number of validators that will be swapped out at a given time. Note that $k < \frac{1}{3}\frac{n}{m}$

Outputs:

1) We set $k = log\frac{n}{m}$
2) for each shard j,
    a) we derive a seed $H(j \| rnd_e)$ to compute a permutation $\pi_{j,e}$ of the shard's validators and specify the permutation of the batches of size k.
    b) we compute another seed $H(0 \| rnd_e)$ to permute the new validators who joined in epoch e and to define the order in which they will do so in batches of size k.

3) After defining the random permutation, each batch waits Δbefore starting the bootstrap process in order to spread the load on the system.
4) When a validator is ready, it requests the shard's leader to let itself swapped in

---

5.5 Analysis

Security Theorem 2: *We ensured safety of BTF consensus in each shard transition.*

Proof: For each shard during the transition phase, there are always at least $\frac{2}{3}\frac{n}{m}$ validators willing to participate in the consensus within each shard, i.e., no quorum of malicious nodes ever exists. Moreover, since we use the epoch's randomness rnde to pick the permutation of the batches, we keep the shards' configurations a moving target for an adaptive adversary. Finally, as long as there are $\frac{2}{3}\frac{n}{m}$ honest and up-to-date validators, liveness is guaranteed whereas if this quorum is breached during transition (the new batch of honest validators has not yet updated) the liveness is lost only temporarily, until the new validators update. ■

6 Inter-Shard Transactions

To support inter-shard transactions as our system needs interoperability for any value transfer(transactions are not tied to a specific institution, state or currency), we present an inter-shard atomic commit protocol that guarantees transaction atomicity cross shards.

A naive approach for handling inter-shard transactions is to concurrently send a transaction to multiple shards for processing, and some shards might commit the transaction while some might abort. In this case, the UTXOs or Contracts at the shard who accepted the transactions are lost as inter-shard states are inconsistent and there is no good way to roll back a half-committed transaction.

To tackle this challenge, we propose a novel Byzantine Shard Atomic Commit (Atomix) protocol for atomically processing transactions across shards such that each single transaction is either committed or aborted. Our system needs to ensure consistency of inter-shard transactions and prevent double spending. The atomic commit protocol improves the naive approach with a simple three-step process.

We will first present Atomix in the UTXO state model and then extend to Smart Contracts model, which enables the following efficient three-step protocol.

6.1 Algorithm - Byzantine Shard Atomic Commit

Algorithm 5: **Byzantine Shard Atomic Commit (Atomix)**

Inputs:

1) crossTX is the Cross-shard transaction
2) ISs is the Input shards

3) OSs is the Output shards

Output:

1) **Initialize**
    a) client creates a crossTX whose inputs spend UTXOs of some ISs and whose outputs create new UTXOs in some output shards OSs
    b) The client gossips the cross-TX and it eventually reaches all ISs
2) **Lock Phase**
    a) Each IS leader validates the transaction within its shard to decide whether the inputs can be spent
    b) If crossTX is verified, the leader marks within the state that the inputs are spent, record the full transaction in the shard's blockchain and finally gossips

a proof-of-acceptance, a signed Merkle proof against the block header where the transaction is included.
    c) If crossTX is not accepted, the leader creates an proof-of-rejection (a special bit indicates the acceptance or rejection).
    d) After the IS validations, the client can use each IS blockchain to verify its proofs and that the transaction was locked.
    e) After all ISs have processed the lock request, the client holds enough proofs to either commit the transaction or abort it and reclaim any locked funds.
3) **Commit-or-abort Phase**
    a) If all IS leaders issued proofs-of-acceptance, then the respective transaction can be committed. An IS leader creates and gossips an unlock-to-commit transaction that consists of the lock transaction and a proof-of-acceptance for each input UTXO. In turn, each involved OS validates the transaction and includes it in the next block of its blockchain in order to update the state and enable the expenditure of the new funds.
    b) If at least one IS issued a proof-of-rejection, then the transaction needs to abort. In order to reclaim the funds locked in the previous phase, an IS leader gossip an unlock-to-abort transaction that includes one proof-of-rejection to request all ISs to unlock that particular transaction. When receiving this request, the IS leaders mark the original UTXOs as spendable again.

6.2 Analysis

Security Theorem 3: *Byzantine Shard Atomic Commit ensured consistency of transactions between shards: (a) all shards always devotedly process valid transaction; (b) if all input*

shards issue proof-of-acceptances, then each output shard unlocks to commit; (c) if at least one input shard issues a proof-of-rejection, then all input shards unlocks to abort; and (d) if at least one input shard issues a proof-of-rejection, then no output shard unlocks to commit.

Proof: Recall that our assumptions are (1) shards eventually receive all messages (2) shards are collectively honest (3) shards can reach BFT-consensus. In atomic commit protocol, each cross-shard transaction eventually either commits or aborts. And this is based on (a) each input shard returns exactly one response - either a proof-of-acceptance or a proof-of-rejection. Therefore if a client has the required number of proofs, then the client either only holds proofs-of-acceptance or not, but not both simultaneously. If the client holds proofs-of-acceptance, this allows the transaction to be committed as (b) holds. If the client doesn't holds proofs-of-acceptance, this forces the transaction to be aborted as

(c) and (d) hold. In atomic commit protocol, no cross-TX can be spent more than once. As illustrated above, cross-shard transactions are atomic and are assigned to designated shards who are solely responsible for them. According to (a) the assigned shards do not process a transaction more than once and no other shard attempts to unlock to commit. If a transaction cannot be committed, then the locked funds can be reclaimed. If a transaction cannot be committed, then there must exist at least one proof-of-rejection issued by an input shard, therefore (c) must hold. After all input shards unlock to abort, the funds is unlocked and available again. ■

Atomix presents a simple and powerful protocol where the shards employes minimal processing logic and there is no direct shard-to-shard communication. We note that stakes are not reclaimed automatically and a client or shard leader needs to send the unlock-to-abort message. Since all necessary information is gossiped, any participants in the system, for example a validator in exchange for a reward, could fill in for the client to initiate an unlock process. We also use a technique to ensure better liveness that we assign the shard of the minimum-value input UTXO to be a coordinator responsible for driving the process of initiating unlock processes. A threat might be that the shard's leader is malicious, and in such case, $f + 1$ validators of the shard need to send the unlock transaction to ensure that all transactions are eventually unlocked.

6.3 Size Optimization

In the Atomic protocol, since proofs for input UTXOs needs to be included, the unlock transactions are bigger than normal transactions. IOSChain relies on Proof-of-Believability (a novel BFT-consensus scheme described in Section 8) for intra shard transactions processing. When a block that contain committed transactions has been verified by the

shard's validators, the validators produce a collective sign whose size is independent of the number of validators. This feature ensures the minimal size of unlock transactions, even though they are verified against the signed block headers of all input UTXOs. If collective signatures are not used, the size of unlock transactions would be unrealistic. For example, a collective signature would only be 80 Bytes for a shard of 100 validators, whereas a regular signature would be 9K Bytes, which is two orders of magnitude larger than the size of a simple transaction.

7 Consensus Mechanism

7.1 Tokens and Motivations

In the IOS system, *IOS Token*, like tokens in other blockchain systems, serve as the medium of exchange for all transactions and commission fees. More importantly, IOS also plays a critical role in calculating a user's believability score. All IOS tokens will be generated in the Genesis Block. In the IOS ecosystem, IOS tokens can be used for:

- **Payment**: Payments for services and goods provided by merchants or other community members.
- **Commission**: Payment to validators as compensation for running smart contracts, processing messages and transactions, and using resources shared by the general ecosystem including but not limited to storage space, computing power, etc. The commission fee incentivizes the validators and prevents malicious users from damaging the interests of the community through excessive deployment of smart contracts.
- **Believability**: IOS tokens will be used in the calculation of a user's believability (explained in the following section).

In addition, as a member of the IOS ecosystem, each user can acquire IOS tokens through validating transactions and contributing resources (e.g. running smart contracts, providing storage space, etc).

As mentioned in previous sections, A major challenge faced by traditional Proof-of-Stake consensus mechanism is the tendency towards centralization. In order to mitigate this risk,

we introduce Servi as both a measurement of users' contributions to the community and a way to encourage members to contribute to the continued development of the IOS community. It has the following attributes:

- **Non-tradable**: Since Servi is not designed as a medium of exchange, Servi can not be traded or exchanged in any way.
- **Self-destructive**: After validating a block, the system will automatically clear the Servi balance owned by the validator, so that nodes with high believability can take turns in validating blocks, thus ensuring a fair block generation process.
- **Self-issuance**: Servi will be generated and deposited to user's account automatically after certain contributions, such as providing community services, evaluating services provided by another party, or making other special contributions.

7.2 Proof-of-Believability

Traditional blockchain systems have an inherent trade-off between safety and throughput, depending on shard size. A system with a large number of smaller shards delivers better performance but provides less resiliency against bad actors, and vice versa. In order to break the trade-off in a way that keeps safety and increases throughput, we propose an innovative Proof-of-Believability (PoB) consensus protocol for IOSChain. PoB guarantees that the nodes are with negligible probability to misbehave, while significantly increasing the transaction throughput by size-one-shard.

The Proof-of-Believability consensus protocol uses an intra-shard Believable-First approach. The protocol divides all validators into two groups, a believable league and a normal league. Believable validators process transactions quickly in the first phase. Afterwards, normal validators sample and verify the transactions in the second phase to provide finality and ensure verifiability. The chance of a node being elected into the believable league is determined by believability score which is calculated by multiple factors (e.g., token balance, contributions to the community, reviews, etc). One with higher believability score is more likely to be elected into the believable league. Believable validators follow the procedures to decide the set of committed transactions and their order, as well as process them in order. Believable validators also form smaller groups - one validator per group. Transactions will be randomly distributed among these believable validators. Consequently, they produce smaller blocks with extremely low latency.

However, it may introduce security problem as only one node is performing the verification. As a result, some corrupted transactions might be committed by misbehaved validators. In order to solve this security problem, we specify a sampling probability p that normal validators will sample transactions and detect inconsistencies. If a validator is detected as misbehaviour, it will lose all the tokens and reputation in the system while the defrauded users will be compensated for any loss. The believable-first approach makes processing transactions extremely fast as only a single (believable) validator is doing the verification and it is unlikely to misbehave.

In the IOS system, the sharding policy file specifies the sizes of the believable and normal league, respectively, and the sampling probability p. Upon the inception of an epoch, all validators will be assigned to shards using the distributed randomness generation protocol. Then, their states will be bootstrapped from the corresponding shard's last Micro State

Blocks (MSB). Depending on the believability score , validators will be assigned to either believable group (small) or the normal group (large) within a shard.

In the first phase, transactions that are processed by the believable league produce optimistically validated blocks. These blocks serve as input for sampling re-validation by the normal league who runs concurrently. The normal league also combines inputs from multiple optimistic processing groups. This could maximize the throughput of the system. If transactions are validated successfully, they will be included in a finalized block, added to the shard's blockchain, and finally included in the MSB. However, when the normal league detects any inconsistency, the corresponding validated transaction would be excluded from the blockchain and the validator who signed the invalid block would be detected and held accountable. We designed the punishment scheme to be powerfully harsh so that the validator has no incentive to misbehave under any circumstances. If a validator is detected as misbehaving, that validator will lose all tokens and reputation in the system and all its previously validated transactions will be re-checked. Given the minimal incentive to be at fault and the quantifiable confidence in the security of validation, clients can achieve real-time processing speed with assurance.

The normal league runs the Byzantine consensus scheme based on ByzCoin [7], because it scales efficiently to thousands of consensus group members. ByzCoin uses collective signature (a.k.a., CoSi) [23], a scalable cryptographic primitive that uses multi-signatures [18], to make traditional consensus algorithms such as PBFT [4] scale. ByzCoin distributes blocks using multicast trees for performance, and falls back to a star topology for fault

tolerance. It ensures that all the honest members of a shard agree on a specific common sequence of actions, despite some malicious nodes are in the shard, while guaranteeing safety and liveness.

To ensure robustness, we use a fall-back scheme in Believable-first protocol. When a shard doesn't have enough believable validators to form the league, due to either temporary downtime or being in the bootstrapping phase of the ecosystem, two-league committees would fall back to one-league. All transactions are directly processed by the normal league following the PBFT consensus protocol.

8 Blockchain Storage Pruning

Another issue current blockchains are facing is the ever-growing size of the blockchain storage, which causes new validators extremely heavy workload for bootstrapping. Blockchains follow the same pattern to store historical data from the beginning. This is a crucial concern for the next-generation high-throughput blockchain systems as the the storage will explode. For example, whereas Bitcoin's blockchain grows at a rate of 100MB per day with a current total size of about 100GB. If it needs to support Visa-level throughput, e.g., 5000 tx/sec and 1000B/tx, that would cause the chain produce over 100 GB per day, which is not affordable under current commercial hardwares and Internet.

To minimize the storage and bootstrapping costs for validators, we introduce Micro State Blocks (MSB). MSB is adopted from classic distributed checkpointing principles, like stable checkpoints in PBFT. It summarizes the full state of a shard's blockchain. We present the MSB generation protocol below.

8.1 Algorithm - MSB Generation Protocol

Algorithm 6: MSB Generation Protocol

Inputs:

1) e is the current epoch
2) i is the current shard

2) j is the current shard

Output: the micro state block $msb_{j,e}$ for j in e

1) At the end of e, the shard leader stores all UTXOs in an ordered Merkle tree [13]
2) Hash the Merkle tree's root, denoted by h, and puts h in header of $msb_{j,e}$
3) Validators run consensus on the header of $msb_{j,e}$, while no regular blocks are pending
4) If consensus is reached that the header of $msb_{j,e}$ is correct, the shard leader stores the approved header in the shard's blockchain thus $msb_{j,e}$ is the genesis block of epoch $e+1$
5) Discard the body of $msb_{j,e-1}$ and keep the regular blocks of e for until end of $e+1$

8.2 Analysis

Since IOSChain state is distributed across multiple shards and each shard's blockchain stores only the MSB headers, a client cannot prove the existence of a past transaction to

another one by providing an inclusion proof to the block. We tackle this issue by moving the responsibility of storing transactions' proofs-of-existence to the clients of IOS. Since we keep latest epoch's blocks, clients can ask the validators of the shard to create proofs-of-existence for transactions validated in epoch e during epoch $e+1$.

Validators create a higher-level Chain (storing these MSBs), that provides skips from an epoch's MSB to another. This MSB-Chain holds the latest MSB in full and all previous MSB headers. This is important as clients that want to verify a past transaction need to have a reference point.

The proof for a given transaction contains:

- Merkle tree inclusion proof to the block that committed transactions in epoch e
- A sequence of block headers from the micro state block msb_{je} at the end of the epoch e to the block that committed transactions.

We further use technique to reduce the size of these proofs by multi-hop backpointers - MSBs may contain several multi-hop backpointers to headers of intermediate regular blocks.

With MSB, bootstrapping new validators or syncing crashed validators up-to-date becomes efficient, as validators start from the last valid MSB and replay only the last part of the

blockchain, instead of replaying the full history from the first block or from the time they crashed. If Bitcoin is deployed on IOSChain, the bandwidth bootstrapping cost would be two orders of magnitude less. This is critical when new shards come in. Due to the random shard assignment mechanism, validators changes shards periodically and need to update.

REFERENCES

[1] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. 2016. Cryptocurrencies Without Proof of Work. In *Lecture Notes in Computer Science*. 142–157.

[2] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. 2014. Proof of Activity. *ACM SIGMETRICS Performance Evaluation Review* 42, 3 (2014), 34–37.

[3] Vitalik Buterin. 2014. Slasher: a punitive proof of stake algorithm. Retrieved January 9, 2018 from https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/

[4] Miguel Oom Temudo de Castro. 2000. *Practical Byzantine Fault Tolerance*.

[5] Nxt Community. Nxt Whitepaper. Retrieved January 9, 2018 from https://bravenewcoin.com/assets/Whitepapers/NxtWhitepaper-v122-rev4.pdf

[6] George Danezis and Sarah Meiklejohn. 2016. Centrally Banked Cryptocurrencies. In *Proceedings 2016 Network and Distributed System Security Symposium*. DOI:https://doi.org/10.14722/ndss.2016.23187

[7] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. 2016. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *25th USENIX Conference on Security Symposium*.

[8] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand. In *Proceedings of the 26th Symposium on Operating Systems Principles - SOSP*

Algorand. In *Proceedings of the 26th Symposium on Operating Systems Principles - SOSP '17*. DOI:https://doi.org/10.1145/3132747.3132757

[9] G Maxwell And. 2015. On Stake and Consensus. Retrieved January 9, 2018 from https://download.wpsoftware.net/bitcoin/pos.pdf

[10] Ian Grigg. EOS - An Introduction. *eos.io*. Retrieved from https://eos.io/documents/EOS_An_Introduction.pdf

[11] J. Kwon. 2014. Tendermint: Consensus without mining. Retrieved January 9, 2018 from http://tendermint.com/docs/tendermint.pdf

[12] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A Secure Sharding Protocol For Open Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*. DOI:https://doi.org/10.1145/2976749.2978389

[13] Ralph C. Merkle. A Certified Digital Signature. In *Lecture Notes in Computer Science*. 218–238.

[14] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. *bitcoin.org*. Retrieved from https://bitcoin.org/bitcoin.pdf

[15] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the Blockchain Protocol in Asynchronous Networks. In *Lecture Notes in Computer Science*. 643–673.

[16] Rafael Pass and Elaine Shi. 2017. The Sleepy Model of Consensus. In *Lecture Notes in*

Computer Science. 380–409.

[17] Phil Daian Rafael Pass. Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proofs of Stake.

[18] C. P. Schnorr. 1991. Efficient signature generation by smart cards. *J. Cryptology* 4, 3 (1991). DOI:https://doi.org/10.1007/bf00196725

[19] Berry Schoenmakers. 1999. A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting. In *Lecture Notes in Computer Science*. 148–164.

[20] Jagdeep Sidhu. 2017. Syscoin: A Peer-to-Peer Electronic Cash System with Blockchain-Based Services for E-Business. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. DOI:https://doi.org/10.1109/icccn.2017.8038518

[21] Sunny King And. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. Retrieved 2012 from https://peercoin.net/assets/paper/peercoin-paper.pdf

[22] Scott Nadal Sunny King. 2012. Peercoin. Retrieved January 9, 2018 from https://peercoin.net/assets/paper/peercoin-paper.pdf

[23] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. 2016. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In *2016 IEEE Symposium on Security and Privacy (SP)*. DOI:https://doi.org/10.1109/sp.2016.38

[24] V Buterin And. 2015. Casper. Retrieved January 9, 2018 from

https://blog.ethereum.org/2015/08/01/ introducing-casper-friendly-ghost/

[25] Gavin Wood. 2018. ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER. *ethereum.github.io/yellowpaper*. Retrieved from https://ethereum.github.io/yellowpaper/paper.pdf

[26] 2017. Steem: An incentivized, blockchain-based, public content platform. *steem.io*. Retrieved from https://steem.io/SteemWhitePaper.pdf

[27] Whitepapers. *bitshares.org*. Retrieved from http://docs.bitshares.org/bitshares/papers/

[28] Proof of stake instead of proof of work. Retrieved January 9, 2018 from https://bitcointalk.org/index.php?topic=27787.0