Received January 8, 2021, accepted January 9, 2021, date of publication January 14, 2021, date of current version January 25, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3051796

Concordia: A Streamlined Consensus Protocol for Blockchain Networks

CARLOS SANTIAGO, SHUYANG REN, CHOONHWA LEE, AND MINSOO RYU
Department of Computer Science, Hanyang University, Seoul 04763, South Korea

Corresponding author: Choonhwa Lee (lee@hanyang.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIT) under Grant 2020R1A2B5B01001758, and in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) Grant funded by the Korean Government (MSIT) (A Delegated Byzantine-Tolerant Consensus Algorithm for Blockchain Scalability) under Grant 2019-0-00458.

ABSTRACT In this paper, we present a novel Byzantine fault-tolerant consensus protocol for sharded blockchain networks that does not rely on expensive leader-driven communication. The proposed protocol selects a single block proposer at a time and uses threshold signatures as a voting mechanism to confirm the validity of the proposed block. By using a gossip-like communication scheme, each node can collect and recover the group signature within $O(\log N)$ steps. With only one block proposer per consensus round, there is no possibility of conflicting blocks and resultant forks. Therefore, our consensus protocol requires only one round of one-way communication to achieve finality for each block. Our protocol guarantees safety and liveness while tolerating up to f faulty participants among $2f + 1$ nodes. Our performance study shows that the proposed protocol enables hundreds of nodes to participate in the agreement process, and can finalize large blocks in approximately 10 seconds.

INDEX TERMS Blockchain, Byzantine fault tolerance, consensus, transaction processing.

I. INTRODUCTION

In recent years, the popularity of blockchain technology has increased at an unprecedented rate. This surge is partially caused by the boom of cryptocurrencies such as Bitcoin [1], Ethereum [2], and Ripple [3]. Although cryptocurrency is currently one of the most important use cases, blockchains can also be applied to supply chain management, healthcare, smart factories, and digital content management [4]. Companies and developers are coming up with different use cases and ideas wherein the distributed ledger technology could provide a viable solution to key fundamental problems of data immutability, replication, and security.

A consensus protocol is an important foundation on which to build blockchain systems. With an adequate consensus scheme in place, a blockchain system should be able to tolerate some faults without disturbing the process of reaching consensus on new blocks. Depending on the use cases, two approaches for blockchains exist: permissioned and permissionless. Nodes in a permissioned blockchain system require prior approval to join the network, whereas a permissionless blockchain allows anyone to participate in the network.

The associate editor coordinating the review of this manuscript and approving it for publication was Derek Abbott.

Several consensus protocols have been proposed in recent years. The most widely-used consensus scheme for permissionless blockchains is proof of work (PoW), which offers good scalability to suit the open membership requirement of permissionless blockchains such as Bitcoin and Ethereum. By contrast, Byzantine fault tolerance (BFT) protocols are the technology of choice for permissioned blockchains. Several protocols have been proposed in recent years to overcome the performance and scalability limitations of previous BFT-based protocols, while maintaining resilience and security guarantees [5], [6]. Some of the most prominent include Zyzzyva [7], AZyzzyva [8], SBFT [9], and Tendermint [10].

Despite recent drastic advancements, blockchain technology still faces several technical challenges to realize its initial vision, including the attainment of a high transaction processing performance necessary for real-world applications. This problem can be mitigated by using sharded consensus schemes, allowing independent transactions to be distributed across a set of shards and processed in parallel [11]. Agreement within a shard is locally managed via an intra-shard consensus protocol, whereas global consensus is achieved by an inter-shard protocol. Thus, sharding-based approaches can achieve a high-performance standard of several thousands of transactions per second (TPS).

This paper presents a new protocol used for intra-shard consensus as part of our long-term efforts to devise a high-performance sharding protocol. Our ultimate goal of developing a high-throughput sharded blockchain technology is carefully considered when designing the first phase of our novel intra-shard consensus protocol. To ensure the network scalability of blockchain systems, individual shards should be able to host a given population. State-of-the-art technologies, such as Elastico [12], OmniLedger [13], RapidChain [14], Harmony [15], and Monoxide [16], typically target the shard size of hundreds of nodes. Therefore, our intra-shard protocol is designed to accommodate several hundreds of nodes per shard. A high adversarial power tolerance per shard, together with an adequate reorganization protocol, are key ingredients to ensure the security of the overall sharded network. Moreover, we aim to provide low block confirmation latency.

We propose Concordia, a BFT protocol for sharded blockchain systems that is geared towards streamlined processing of blocks. The operation of the consensus protocol is organized in rounds. To create a block for each round, a single block proposer is selected in a random and non-interactive manner, meaning that consensus participants can determine the identity of the next block proposer without communicating with one another. Additionally, the protocol uses a threshold Boneh-Lynn-Shacham (BLS) signature scheme [17] as the voting mechanism for the network to agree on a valid block. To minimize the finalization time, each block is finalized by only one round of signing. Correct nodes can then obtain sufficient signature shares to validate a block within O(log$N$) steps and combine them into a single unique group signature, requiring only one signature validation to verify the block.

We make the following key contributions:

- We allow hundreds of nodes to participate in a consensus process by using a threshold BLS signature combined with a gossip communication method.
- The process of block finalization can be completed by only one round of signature share exchanges. Thus, our protocol provides strong consistency with a transaction confirmation latency of approximately 10 seconds.
- Our protocol guarantees safety and liveness to the correct nodes as long as the adversary controls less than 50% of the voting power.
- The operation of our protocol is made smooth and seamless by eliminating costly view changes.

The reminder of this paper is organized as follows. In the next section, we briefly discuss the technological background of our work. Section III presents the overall design of Concordia with a detailed description of its key components. After presenting the basic security analysis of our protocol in Section IV, we report and discuss our performance evaluation results in Section V. Section VI summarizes the related and future works. Finally, Section VII draws the conclusion for this paper.

II. BACKGROUND

In this section, we review the technical background for our proposed protocol, including sharded consensus networks, fault-tolerance models, and signature aggregation techniques.

A. SHARDING

The most promising solution to the scalability problems of blockchain systems is sharding. With this approach, the network is divided into smaller groups (or shards), allowing each of them to validate independent blocks. Thus, the process of validating transactions is parallelized, and the throughput of the system increases proportionally with the number of shards, making the system scalable. Although this promising mechanism appears to solve most current protocol problems, it introduces additional issues that require completely new approaches and solutions. The major requirements to build a sharding protocol are:

- A method to create shards and reorganize nodes across them, to ensure that the security of each shard is never compromised, as well as a Sybil-resistant method to generate identities.
- A consensus protocol to confirm blocks in parallel on each shard, also referred to as an intra-shard consensus protocol.
- A method to validate transactions that involve information stored on ledgers managed by different shards. These are referred to as "cross-shard transactions."
- A communication protocol to propagate blocks and transactions between nodes of the same shard and across shards.

All modules of the protocol must work in harmony with others to obtain a reliable, fast, and secure protocol, capable of scaling up the throughput of blockchain systems and open a new broad range of applications. The work presented in this paper mainly focuses on the creation of an intra-shard consensus protocol to propose a solution to one of the key components of sharding blockchain consensus protocols. The remaining aspects will addressed in the future to eventually create a high-performance blockchain sharding consensus protocol built with Concordia.

B. FAULT TOLERANCE

Permissioned blockchains employ distributed consensus protocols that are based on an adversary model different from that of mining-based consensus protocols. An attacker might be able to prevent the network from reaching consensus by disrupting communications among the nodes. Lamport [18] established that, to tolerate f non-Byzantine faults, the network should accommodate at least $2f + 1$ replicas. Castro and Liskov [19] showed that a total of $3f + 1$ replicas are needed to be able to tolerate Byzantine faults. These thresholds have been proven, and together with the network model, they establish the rules and limitations of distributed consensus systems. The practical BFT (PBFT) protocol relies on a weak synchronous network for liveness; however, when considering practical deployments, its adversary model is considered as too strong, because malicious Byzantine fault nodes rarely cause the entire network to break down. Thus, a cross-fault-tolerant (XFT) protocol [20] was proposed based on a more realistic model that provides safety and liveness to the system. Byzantine faults are allowed to occur in the network, while a majority of the nodes are able to communicate synchronously.

C. THRESHOLD SIGNATURE SCHEME FOR SCALIBILITY

Threshold signature schemes can help consensus protocols scale for larger-sized networks. Several distinct partial signatures can be aggregated into a single signature so that multiple proofs can be verified in a single operation.

The BLS signature scheme [21] utilizes the gap Diffie-Hellman (GDH) groups, where the computational Diffie-Hellman problem is hard, but the decisional Diffie–Hellman problem is easy. The BLS signature scheme provides the signature generation algorithm and the verification based on a GDH group, G_1, of prime order p_1 and a generator, g_1. Generator g_1, order p_1, and a description of hash function H are together considered as the global information. With a key pair $(\mathrm{sk}_i, \mathrm{pk}_i)$ for each party, signature σ_i of message M is computed as $\mathrm{H}(M)^{\mathrm{sk}_i}$. To verify the signature, the verifier must check whether $(g_1, \mathrm{pk}_i, \mathrm{H}(M), \sigma_i)$ is a valid Diffie-Hellman tuple.

Derived from BLS, the threshold BLS signature scheme [17] works in a non-interactive and leaderless way since the signature share generation is congruent with BLS, and the result of group signature recovery is a process of calculating without interaction. The threshold BLS signature scheme consists of three algorithms: a key generation algorithm, a signature generation algorithm, and a verification algorithm. The key generation algorithm provides a way to distribute the necessary keys to participants using a distributed key generation (DKG) protocol. The signature generation algorithm contains a signature share generation protocol, which is the same as the BLS signature scheme and a group signature recovery protocol, where the output is the ''Lagrange interpolation'' of the signature shares. The verification algorithm is identical to the BLS verification algorithm, but using the group public key generated by the DKG algorithm.

The key generation algorithm of the threshold BLS signature scheme we utilized is a discrete log-based DKG protocol [22]. This protocol outputs a private-public key pair $(\mathrm{sk}_i, \mathrm{pk}_i)$ for each party, and a group public key, pk, to verify the generated group signature. To run a discrete log-based DKG, a cyclic group, G_2, of prime order p_2 with a particular generator, g_2, is needed. Each node selects a random polynomial, z_i, and distributes the polynomial among n nodes. The secret shares of each node s_i used to recover the unique value s, are a pre-decided linear combination of z_i. The group public key, ${g_2}^s$, will be used to verify the group signature. The execution of the DKG protocol takes time; however, the protocol does not run frequently.

To accommodate a larger number of nodes into a consensus network, we combine a t-of-n BLS threshold signature scheme with a gossip-like communication protocol. Consequently, our consensus protocol reduces the communication complexity substantially. As presented in the following section, the aggregated signature is also utilized as the key component for the randomness generation in our protocol, serving as a verifiable random function to randomly select a block proposer. Since the signature aggregation can be performed by any participant, the responsibility of the block proposer is limited to proposing a block. It is noteworthy that, unlike in our proposal, leaders in other consensus protocols are typically tasked with driving the entire communication and consensus process.

III. CONCORDIA CONSENSUS PROTOCOL

We designed Concordia to be a protocol that streamlines the process of reaching consensus on new blocks. We achieve this goal by removing unnecessary complexity derived from unrealistic assumptions, and dispensing with expensive fault detection and view-change mechanisms, in exchange for a straightforward protocol that achieves great performance while ensuring security. We realize this vision by designing a new accelerated voting mechanism using aggregated signatures integrated with the gossip communication protocol, together with a random block proposer selection method that inherently prevents the formation of forks.

To create a highly efficient protocol, all steps and modules of the protocol work together in a seamless manner. The verification process is conducted by relaying signature shares using a gossip-based messaging protocol. As the signature shares for a block proposal propagate through the network, they are recovered into the group signature that serves as the block confirmation. The group signature is then utilized as a randomness source to select a block proposer for the next round. Thus, all components of our protocol are coherently integrated with each another.

Concordia is designed to be used in networks where all nodes are connected by a broadcast channel as well as reliable point-to-point channels. The protocol provides safety and liveness guarantees as long as more than $f + 1$ nodes are correct and synchronous, where message exchanges between two correct nodes are delivered and processed within delay Δ. Additionally, we assume that the adversary is computationally bounded. Hence, it cannot break cryptography algorithms' security, including digital signatures and hash functions.

A. OVERALL ARCHITECTURE

In general, permissioned consensus protocols can yield higher transaction throughput than their permissionless counterparts, but are limited to small-sized networks, owing to their high communication complexity. Having small networks makes the system more vulnerable to adversarial attacks such as Sybil or targeted attacks. These factors are the opposite in the case of permissionless blockchain networks.

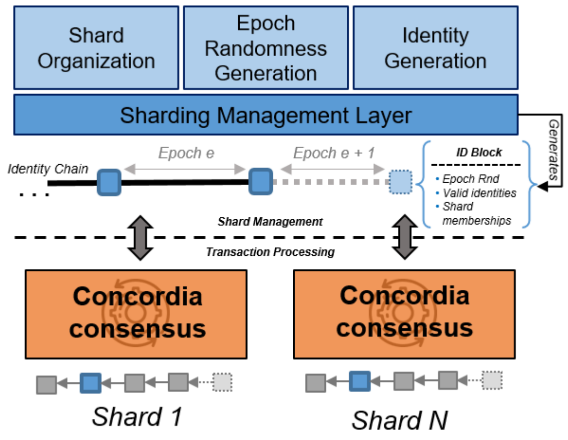


FIGURE 1. Sharding-based consensus network architecture.

To reap the best of both worlds, hybrid consensus has been proposed by which consensus committee members are randomly selected among permissionless network nodes to run a fast permissioned Byzantine agreement in each committee [23]. Various methods have been explored to establish Sybil-attack-resistant identities, including a PoW scheme or a cryptographic sortition function [12], [13], [24]–[27]. Thus, the fast performance of permissioned consensus protocols could be achieved without sacrificing the security and scalability of permissionless open-membership networks. A similar approach is adopted in our work.

As shown in Fig. 1, Concordia is designed to work in a permissioned environment that can be enabled by a management layer that splits the network into synchronous committees, forming parallel shards. The network is organized into a set of consensus committees responsible for a segment of the ledger where they process transactions independently using Concordia. The system works with an additional management layer that produces *identity blocks*, that contain the identities of all participants and the consensus shard to which they are assigned to. When an identity block is created, it is relayed across the network and is appended to each shard's chain by committing it together with the immediate next transaction block. This event marks the beginning of a new epoch for the entire system, with following transaction blocks being derived from the fresh epoch randomness. An epoch is a time period during which shard memberships are fixed, and nodes cannot leave or join consensus shards. Thus, the sharding management layer allows shards to implement a permissioned consensus blockchain like Concordia, whereas the overall system is open to any participant that can submit a join request to the identity generation module.

A new node that wants to join the system, submits a Sybil-attack-resistant proof, such as a PoW puzzle or a proof of locked funds, to the sharding management layer. If the submitted proof is valid, the identity is added to the following identity block. The shard assignation and reorganization is performed in a random and progressive manner based on the randomness generated on the previous epoch by the sharding management layer. In other words, only a portion of nodes are commanded to migrate to other shards to prevent nodes from having to download new ledgers frequently. There are several works on dynamic committee formation. In Elastico [12], committee membership is based on the result of a PoW puzzle, and in RapidChain [14], there is a reference committee that randomly maps identities to shards using the Cuckoo rule [28] to protect against slowly adaptive Byzantine adversary.

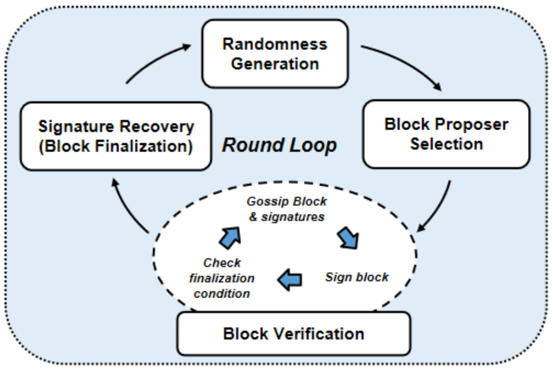

Randomness Generation
Signature Recovery (Block Finalization)
Round Loop
Block Proposer Selection
Gossip Block & signatures
Check finalization condition
Sign block
Block Verification

FIGURE 2. Main components of Concordia.

B. CONCORDIA PROTOCOL

We aim to build a protocol that allows hundreds of nodes to participate in the consensus procedure while offering low finality latency. The consensus protocol proceeds in sequential rounds. Each round, a random block proposer is selected; then, every participant is allowed to vote once, where the signature share of the participant on the block is counted as a vote for the validity of the block. As illustrated in Fig. 2, our protocol consists of four main components.

- A distributed randomness generation scheme based on the threshold signature generated in the immediately previous round.
- A block proposer selection scheme based on the generated random seed.
- An accelerated block verification process embedded with a gossip communication protocol.
- An efficient group signature recovery scheme, where the group signature is the proof of block finalization.

The beginning of the consensus protocol starts with the generation of key shares for all nodes; subsequently, the protocol proceeds in sequential rounds. Each round operates as follows.

1) Nodes independently generate the round randomness based on the collective signature of the last block.
2) Based on the round randomness, a block proposer is selected.
3) The block proposer generates a block and gossips it to all other nodes in the network.
4) As nodes receive the block proposal, they execute the block verification and finalization process by

appending their signature shares to the valid block and gossiping newly received valid signatures.

5) When a node receives enough signatures (more than the threshold), it recovers the group signature as the proof of finalization of the block.
6) Nodes that (a) gather enough shares and recover the final signature or (b) receive the finalized block with the group signature, will gossip one last whisper with the finalized block.
7) When nodes receive or generate the group signature, they compute a new round randomness, triggering the start of the next round.

Algorithm 1 Consensus Round Loop

```
while true do
    // Iteration for round $r$;
    $\mathrm{Rnd}_r = \mathrm{GenerateRoundRandomness}(r, \sigma_{r-1})$;
    $\mathrm{Bpp} = \mathrm{PickBlockProposer}(\mathrm{Rnd}_r)$;
    if $me = Bpp$ then
        $B_r = \mathrm{GenerateBlockProposal}(B_{r-1}, \mathrm{data})$;
        $\mathrm{Gossip}(B_r)$;
    // Block verification and finalization loop;
    while !finalized do
        sigShares, groupSig, $B_r = \mathrm{RcvNewInputs}()$;
        // ** Check if the block is legal;
        if isValid(groupSig) then
            $\sigma_r = \mathrm{groupSig}$;
            $\mathrm{Gossip}(B_r, \sigma_r)$;
            finalized = true;
        else if count(sigShares) $> f + 1$ then
            $\sigma_r = \mathrm{RecoverGroupSig}(\mathrm{sigShares})$;
            $\mathrm{Gossip}(B_r, \sigma_r)$;
            finalized = true;
        else
            sigShares = AppendOwnSignature($B_r$);
            Gossip($B_r$, sigShares);
    $r = r + 1$;
```

The operation of the protocol is also presented in Alg. 1, showing the process that needs to be executed to complete each round of consensus. Further details for every stage are described in the following subsections, where we present our protocol step-by-step. We address the challenge of block proposer determination in addition to block verification and finalization to guarantee the safety of the system. The operation of our protocol is smooth and seamless even in the presence of an adversary.

C. PROTOCOL SETUP

For a node to start processing transactions and validating blocks, it needs a self-generated Rivest-Shamir-Adleman (RSA) key pair that each node submits to the management layer, as well as a private key share to execute the BLS threshold signature scheme. Identities are provided by consulting the identity block generated in each epoch and, using a peer-discovery algorithm, participants in the same committee are able to have network connectivity with each other. To generate the key shares for block validation, the committee needs to run a DKG algorithm to generate the BLS key shares in a decentralized manner.

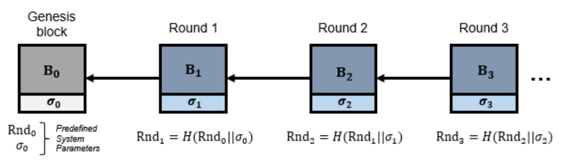

Genesis block
Round 1
Round 2
Round 3
B_0
σ_0
B_1
σ_1
B_2
σ_2
B_3
σ_3
...
Rnd_0
σ_0
Predefined System Parameters
$\mathrm{Rnd}_1 = H(\mathrm{Rnd}_0||\sigma_0)$
$\mathrm{Rnd}_2 = H(\mathrm{Rnd}_1||\sigma_1)$
$\mathrm{Rnd}_3 = H(\mathrm{Rnd}_2||\sigma_2)$

FIGURE 3. Calculation of each round randomness.

We choose the threshold GDH signature scheme [17] for partial signature share generation and aggregated signature reconstruction, which is specifically based on the BLS signature scheme [21]. Our protocol makes use of three relevant functions of the threshold BLS signature scheme: a signature method to generate partial signature shares, a recovery algorithm that reconstructs the full signature from a threshold of signature shares, and a verify function to both validate the signature shares and the full signature. Throughout the paper, the signature recovery algorithm is used interchangeably with signature aggregation. To generate and distribute the necessary keys for the participants, we make use a previous study that allowed a set of n parties to jointly generate a pair of public and private keys without a trusted party [22].

D. DISTRIBUTED RANDOMNESS GENERATION

The distributed randomness generation protocol should enable the participants to jointly produce an output that is verifiable, unbiased, and unpredictable across the system. To satisfy this requirement, the distributed randomness generation is built upon a secure and robust threshold BLS signature scheme, which can tolerate any f malicious parties among $2f + 1$ [17].

When the participants of the consensus procedure in the current round receive the group signature, σ_r, they execute the distributed randomness generation protocol to generate round randomness Rnd_{r+1} for the next round. As shown in Fig. 3, we use the group signature, σ_r, of the current round as a source of randomness, and combine it with the previous round randomness, Rnd_r. For simplicity, the round randomness for the very first round, denoted as Rnd_0, is set to be the hash value of the "genesis block". In subsequent rounds, Rnd_{r+1} is calculated as the hash value of Rnd_r concatenated with the group signature, σ_r. The calculation is performed as in (1).

$$\mathrm{Rnd}_{r+1} = \mathrm{H}(\mathrm{Rnd}_r \mid\mid \sigma_r) \tag{1}$$

Concordia is robust, meaning that an adversary cannot prevent the protocol from reaching consensus. Based on the assumption that the majority of the nodes behave synchronously, the required threshold of the signature scheme is always satisfied. Thus, the recovery of the group signature

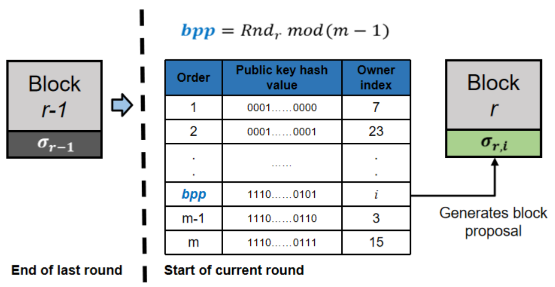


FIGURE 4. Selection of a block proposer for round *r*.

for each round is unpreventable. The group signature will be generated by all correct parties or received with very high probability through the gossip protocol.

Furthermore, the output of the protocol is unpredictable and unbiasable, whereas the result of the generation is consistent. Rnd_r is a pre-known parameter; however, the group signature, σ_r, is not revealed until the end of round r, making it impossible to predict ahead of time. Even if one of the participants finishes before the others and is the first to recover σ_r, the node cannot temper the result of the recovery. The generation of Rnd_{r+1} is computed based on verifiable common inputs, guaranteeing that the output is always consistent.

Another benefit of this method is that upon receiving the valid group signature, σ_r, the generation of Rnd_{r+1} is non-interactive, enabling nodes to swiftly start the following round with no further communication.

E. BLOCK PROPOSER SELECTION

After execution of the key generation protocol, each node has a public-private key pair, $< \text{PK}_i, \text{SK}_i >$, to sign and verify messages in which the public key is known by all nodes in the system. Before participating in the consensus protocol, the nodes exchange the public keys as the identities of participants. When a node gets the list of public keys, the identities are ordered based on the public key hash value from low to high. Thus, as illustrated in Fig. 4, all participants have the same view of the list of the public keys with the owner's index.

The legal block proposer is chosen from the node list. The participants determine the block proposer position, Bpp, in the public key ordered list based on the round randomness, Rnd_r. The block proposer position, Bpp, can be calculated using (2), where m is the total number of nodes.

$$\text{Bpp} = \text{Rnd}_\text{r} \bmod (m-1) \qquad (2)$$

The selection of the block proposer is based on Rnd_r which, as introduced before, is unbiased and unpredictable. As a result, the block proposer is determined in a secure and random manner. Additionally, similar to how the round randomness is generated, the settlement of the block proposer is calculated by each node independently. Since the inputs for the Bpp calculation are identical for all nodes participating in the consensus, the result is consistent and can be easily verified with the block proposer's signature attached on the proposed block.

The node whose identity corresponds to Bpp is selected as the legal block proposer for the current round. The block proposer gathers transactions to generate a block represented by tuple $(r, p, \sigma_{r,Bpp}, \text{root}, d)$, where r is the round number, p is the hash pointer to the previous block, root is the Merkle tree root of the transaction list, and d is the transaction data. The block proposer then proceeds and gossips the block with its signature share, $\sigma_{r,Bpp}$, appended. The process of block generation is illustrated in Fig. 4.

F. BLOCK VERIFICATION AND FINALIZATION PROCESS

We employ a randomized gossip protocol to propagate blocks and signatures throughout the network. After block generation, the block proposer randomly chooses $\log(N)$ participants to which to send the block with its signature share attached. When a node receives a new block proposal, it checks $\sigma_{r,Bpp}$ to determine whether the block is legitimately generated by the chosen block proposer. Additionally, nodes evaluate the correctness of the following components to asses the validity of the block.

- Previous pointer p: The previous pointer has to be equal to the hash of the immediate predecessor block, which was finalized during the previous round.
- Merkle root root: The Merkle root is verified as the abstract of the transaction list.
- Data d: The transactions are verified against current user states.
- Signature shares $\sigma_{r,i}$: Block proposals contain a list of signature shares vouching for the validity of the block. Each signature share's validity is checked upon reception.

If all conditions are met, a node would then gossip out the received block with all previously gathered signature shares and the node's own share. As shown in Alg. 1, the process of verifying the block repeats until the block collects enough number (threshold) of valid approvals. When the block obtains $f+1$ signatures, it is considered to be verified. The signature shares are then recovered into a single group signature, which is the proof of finalization of the block. The process of block verification is also depicted in Fig. 5.

As mentioned previously, the signature shares propagate with the block. Thus, to gather enough signature shares, our finalization protocol requires only one round of the signing process, which relies on efficient one-way communication. Additionally, the recovery of the group signature could be done by any participant. Therefore, the block finalization protocol runs in a completely decentralized and leaderless manner, reducing the probability of single node failures.

The group signature, σ_r, is sufficient to be the proof of the finalization of the block, not requiring further message exchanges. The successfully recovered group signature, σ_r, proves that a sufficient number of participants signed the block, and can be easily verified with the corresponding group public key. When σ_r is gossiped out, σ_r is received

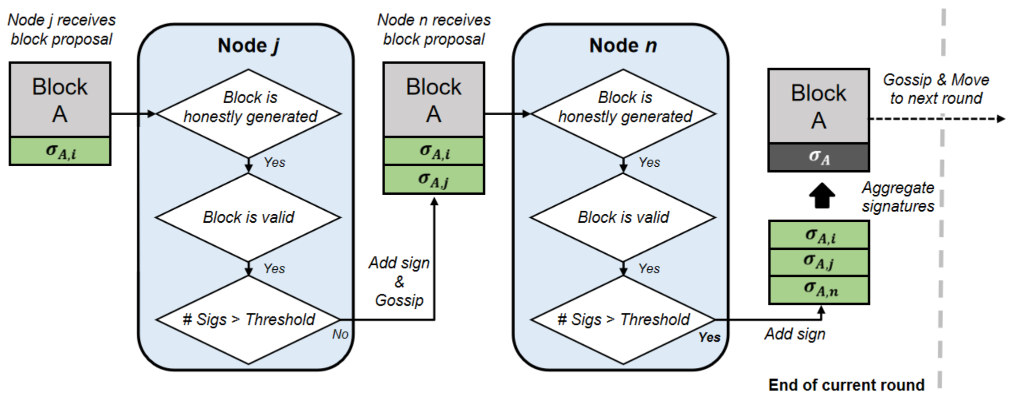


FIGURE 5. Block verification and finalization process.

by all correct nodes within Δ. Thus, σ_r is treated as the proof of finalization of the block. Moreover, because correct participants only sign one block per round, only one block passes the verification and finalization process. As a result, the protocol can provide consensus finality to the proposed block for each round, inherently preventing the creation of forks.

G. PROTOCOL OPERATION UNDER FAULTS

To guarantee the continuous operation of the consensus protocol, participants are forced to proceed to the following round in case of a fault. Different from PBFT, where the leader (and block proposer) remains fixed unless it fails, our protocol forces a block proposer to change proactively, triggered by the group signature.

In our protocol, a round can only have two possible outputs: a valid finalized block or a finalized empty block, which has the same content as an ordinary block but without containing any transaction. A valid block is finalized only when (1) the block proposer acts correctly proposing a valid block and (2) a majority of benevolent validators receive it and sign it. If condition (1) is not met, meaning that the block proposer sends an invalid block, it triggers other nodes to determine the round as invalid, signing and gossiping an empty block together with the invalid proposed block as a proof of misbehaviour of the current block proposer. In contrast, if condition (2) is not satisfied, it means that either there is a network problem, or a portion of validators have declined to sign the block. In this case, validators timeout and propose an empty block. Other nodes that timeout would also sign the same empty block, eventually finalizing it and triggering the start of the following round. Note that validators sign an empty block strictly only when they timeout or if they have a proof that the block proposer sent an invalid block.

Thus, we can guarantee that enough signature shares are gathered, either for a valid block or an empty block, and recovered into a group signature that will be used as the source of randomness for the following round. As a result, the responsibility of generating the next block will be assigned to another node, making it impossible for the adversary to prevent the continuous operation of the protocol.

IV. SECURITY ANALYSIS

In this section, we conduct a security analysis to prove that our protocol provides safety and liveness to the blockchain system. With a majority of the nodes being correct and synchronous, our Concordia protocol guarantees:

- *Safety:* all correct nodes commit on the same identical block for a particular consensus round.
- *Liveness:* all correct nodes eventually commit a block and generate the round randomness for the following round.

As introduced at the beginning of Section III, Concordia works in a context where validators have to join a consensus committee by submitting a Sybil-attack-resistant proof to the management layer, thereby ensuring a random organization of the groups. Consequently, consensus conditions among shards are akin to those present in a permissioned network, inherently preventing the Sybil attack. Moreover, in the improbable event of a prolonged stall of a consensus committee due to quorum impossibility caused by the failure of a majority of benevolent nodes or a denial-of-service attack, the stalling time is upper-bounded by the epoch time. Furthermore, the system architecture allows for additional mechanisms to detect committee failures and trigger committee reorganization that will be further considered in our future work. Nevertheless, we argue in this section that Concordia can guarantee security in all circumstances and liveness under eventual synchrony, making it a reliable and trustworthy protocol set for real-world applications.

A. CONSENSUS SAFETY

To guarantee safety, the protocol needs a quorum of $f + 1$ nodes to fulfill the requirement of the threshold signature.

It is known that the threshold signature scheme is secure, which means that the output of the scheme is unforgeable and robust [17]. The threshold signature scheme has two important properties:

- Uniqueness: the scheme allows any subset of $f + 1$ parties to jointly create the group signature. In other words, the result of the partial signature shares recovery is always the same, regardless of whom the signature shares belong to.
- Verifiability: the group signature can be verified by anyone using the unique fixed public key. The public key is created and distributed by the network's key generation protocol at the very beginning of the consensus process.

Our consensus protocol does not rely on the correctness of the block proposer to ensure safety. A malicious block proposer might propose contradicting block proposals to perform an attack against the network.

Claim 1: All correct nodes will commit on the same block, even if a malicious block proposer generates contradicting blocks.

To guarantee safety in such a situation, we allow correct nodes to sign only once per round. Thus, either only one of the blocks or none of the contradicting blocks obtains enough partial signature shares to recover its group signature. If the correct nodes commit on one of the blocks, the block is finalized. If none of the blocks is finalized, the correct nodes commit on an empty block.

Claim 2: If correct node i commits on block B, then all other correct nodes commit on B.

If node i signs B, meaning that B is valid, all other correct nodes also sign on B. Thus, B will be able to obtain enough signature shares to recover the group signature, which could be verified by any party using the unique fixed public key. Thus, the group signature is enough to be the proof of finalization of the block. Once the group signature is disseminated throughout the network via gossiping, all correct nodes receive the group signature and commit on B.

B. CONSENSUS LIVENESS

For liveness, we need a quorum of $f + 1$ correct nodes that behave synchronously at any time, guaranteeing that two subsets of synchronous nodes will always have at least one joint member who participates in the agreement to prevent the protocol from stalling.

Claim 3: Supposing that C is the size of a subset of correct and synchronous nodes, that take part in the consensus in a certain round. Liveness is guaranteed as long as $C \geq f + 1$.

With a quorum of $f + 1$ out of $2f + 1$ nodes, two quorums always intersect in at least one correct node. Thus, if the formation of the synchronous subset changes after gossiping a valid block B, at least one of the correct nodes gossips block B to the new subset of nodes. As a result, all correct nodes obtain B and start the block verification process. Similarly, if the correct subset changes after some nodes gossip out the signature shares attached to B, at least one correct node sends out B with the signature share attachments that could be aggregated into the group signature. Owing to the uniqueness property of the threshold signature scheme, the result of the group signature is consistent regardless of who signs. If the subset membership changes after the gossiping of the group signature, at least one correct node will gossip out the group signature to the new subset. Then, all correct nodes should acquire the group signature and commit on block B.

Claim 4: If node i is correct, it terminates and obtains the randomness seed for the next round even with f faulty nodes present in the system.

Liveness does not rely on a correct block proposer. In other words, liveness can be guaranteed under the influence of adversaries. They could collude with one another to launch three types of attacks: intentionally not proposing a valid block, refusing to exchange signature shares with other nodes, and stopping the propagation of received messages. If a malicious block proposer does not generate any block at all, the correct nodes timeout and commit on an empty block. If a malicious proposer generates an invalid block, the invalid block would not pass the validity check. Therefore, correct nodes also revert to an empty block and start the round randomness generation, once the group signature is obtained. If f malicious nodes refuse to exchange their signature shares, the remaining $f + 1$ correct nodes would always be able to surpass the threshold and recover the group signature. It is possible for the correct nodes to fail to recover the group signature if the adversary prevents messages from being propagated, which would be a rare event, given a proper neighbor size of the gossip protocol. Nevertheless, in that unlikely situation, correct nodes would also eventually timeout and commit on an empty block.

C. RANDOMNESS CREATION

Our randomness creation protocol is based on a robust threshold signature scheme. As long as $f + 1$ nodes participate in the signature aggregation, a malicious node cannot prevent the network from generating a valid group signature. With the robust randomness seed, the adversary cannot manipulate the process of the block proposer selection. Thus, it is guaranteed that the block proposer is selected in an unpredictable and unbiased way. However, it does not guarantee that the selected block proposer is always honest. An adversary might have at most $1/2$ chance per round to be selected as the block proposer. Consequently, the probability that an adversary controls n consecutive block proposer selections is upper-bounded by the following equation:

$$P[X \geq n] = 1/2^n < 10^{-\lambda} \tag{3}$$

For instance, given $\lambda = 6$, the adversary can control at most 20 consecutive rounds, meaning that the probability of the adversary controlling more than 20 rounds is less than 10^{-6}, which can be considered negligible.

D. SECURITY OF GOSSIP

A malicious node could also reject to gossip out any message. Consequently, if a valid block proposer randomly selects $\log(n)$ malicious neighbors, the adversary can deny the block reaching correct nodes. Using the hypergeometric probability cumulative distribution function (4), we can model this scenario and calculate the probability of a node selecting only

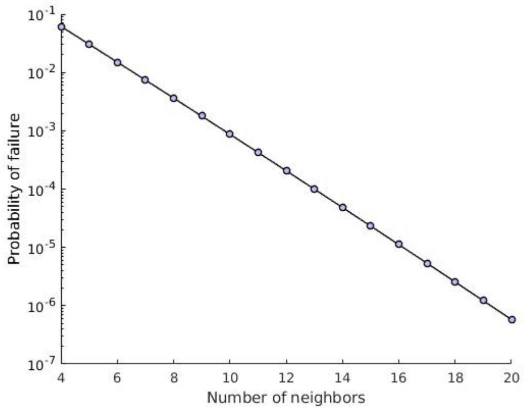


FIGURE 6. Complement of hypergeometric CDF with total population size of $M = 500$ and a Byzantine node population of $K = M/2$.

Byzantine neighbors from a pool of nodes containing half malicious nodes.

$$p = \mathrm{F}(x) = \sum_{i=0}^{x} \frac{\binom{K}{i}\binom{M-K}{N-i}}{\binom{M}{N}} \tag{4}$$

Considering a neighbor selection size m, $F(m)$ returns the probability of drawing m Byzantine nodes of a possible K Byzantine nodes, when drawing in groups of N size without replacement from a total population of M nodes. Fig. 6 plots the complement of the hypergeometric cumulative distribution function (CDF) for different values of x. The result corroborates that the probability is low enough to avoid problems with security and performance.

In the rare event of a block proposer selecting only malicious nodes as neighbors or a Byzantine node being selected as the block proposer, correct participants eventually timeout, triggering the process of validating an empty block and moving on to the following round.

In summary, the probability of an unsuccessful round is bounded by the adversary fraction in addition to the probability of gossiping failures. Nevertheless, the security of the network can never be violated at the expense of performance. In the next section, we further analyze how the presence of the adversary may affect the overall performance of the protocol.

V. PERFORMANCE EVALUATION

In this section, we evaluate our protocol by performing a set of experiments and analyzing the results. The objective of this evaluation is to assure the viability of our protocol and evaluate its performance.

A. EXPERIMENTAL SETUP

We implemented a prototype of Concordia in Go,[1] with all the essential functionality and characteristics described in this paper. We built on top of the Cothority[2] framework. We used Cothority's networking module to implement the message exchange and handling between hosts, and the simulation module to test the protocols and deploy them on our testbed to perform bigger-scale experiments, while collecting performance metrics. Note that, in our prototype, there are no clients, and the verification of transactions is emulated by adding a delay that grows with the block size. Unless otherwise specified, experiments consider the case where all selected block proposers are benevolent, and behave synchronously with a pre-configured Δ value.

To run the experiments, we used a varying size cluster of computing optimized AWS EC2 c5.2xlarge instances, each having 8 vCPUs and 16 GB RAM. On each of the instances, we ran a virtualized Mininet[3] topology with up to 14 hosts running a node of the protocol.

The virtualized Mininet topology is used to modify the characteristics of the links and emulate realistic network conditions such as delays and bandwidth restrictions. For all the experiments, we established 100 ms propagation delays with 35 Mbps bandwidth links between each host to mimic the conditions of a heterogeneous wide-area network.

To compare the capabilities of Concordia with other relevant works, we also performed experiments with implementations of PBFT, XFT [20], FBFT [15], and ByzCoin. The experiments were conducted using the same testbed under identical restrictions.

B. CONSENSUS LATENCY AND THROUGHPUT

The most relevant metric that we use to evaluate the performance of consensus protocols, is consensus latency: the time it takes to complete a full round of the protocol. In other words, it is the time needed to append a completely new block to the blockchain. This latency is mainly affected by two parameters: the number of nodes participating in the consensus and the block size.

Depending on the block size used and the resulting consensus latency, we can calculate the number of TPS that the system can finalize to measure throughput.

For the first experiment, we created a fixed-size network with 100 consensus nodes running on 10 separate instance machines. We ran the protocol for 10 consecutive rounds with block sizes ranging from 1 KB to 5 MB. It is important to note that, due to the nature of our protocol, some nodes finish a round before others; for all of our experiments we use the same node as the source of our metrics.

As shown in Fig. 7, the consensus latency increases steadily with the block size, achieving a finalization latency of less than 10 seconds for block sizes of up to 1 MB among 250 participants. Owing to the network bandwidth limitation of 35 Mbps on all node links and the propagation delays, gossiping larger blocks introduces overheads that make the consensus latency higher and more unstable. Additionally, detailed results show that block propagation accounts for more than 70% of the overall finalization time. This is a clear bottleneck compared with the other steps of the protocol. Nonetheless, the results demonstrate that, because of the

[1]golang.org
[2]github.com/dedis/cothority
[3]mininet.org

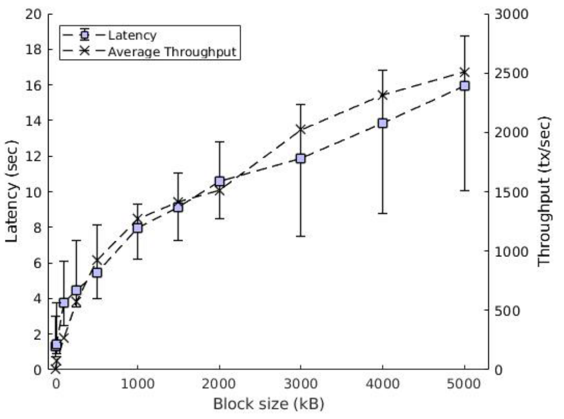


FIGURE 7. Fault-free performance with different block sizes.

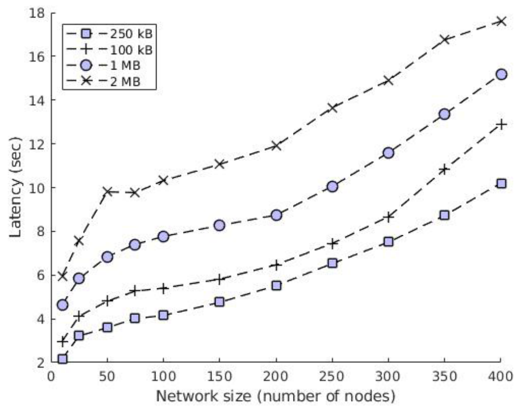


FIGURE 8. Latency vs. network size.

way we implement block propagation, we can handle large block sizes while still achieving reasonable latencies.

The next experiment analyzes how the number of participants in the consensus protocol affects the consensus latency. We progressively increased the number of hosts from 10 up to 400 and replicated the same experiment with four different block sizes. The results are shown in Fig. 8, which plots the average latency of 10 rounds of the consensus protocol. As in the previous experiment, we can see that the latency scales well with the block size. As for the number of participants, we see an almost linear increase of the latency, achieving acceptable values even for network sizes of 400 nodes. Again, by using a gossip-based communication protocol, nodes only need to send blocks to $\log(N)$ nodes, achieving similar block propagation times with the same network bandwidth restrictions. However, as the number of participants increases, the number of signatures needed to confirm a block grows proportionally. This means that all validators need to verify more partial signature shares, and the final signature aggregation process also has a higher threshold. As a result, validators need more computing time to validate signatures on every gossip round, and the final signature aggregation is also slower. Later in this section, we expand on this argument and evaluate the delay introduced by the signature verification and recovery processes.

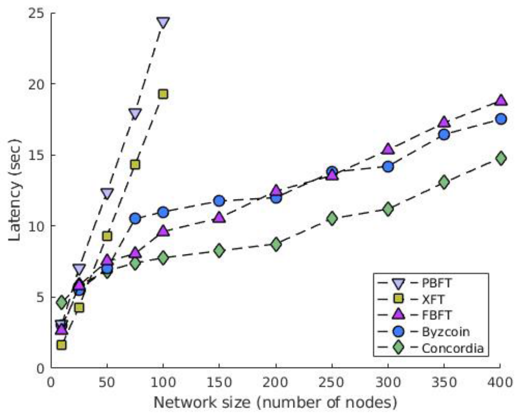


FIGURE 9. Fault-free performance comparison.

Finally, to compare the results of Concordia with other work, Fig. 9 shows the average latency needed to finalize a 1MB block with a varying network size for different protocols.

The comparison includes the results obtained running implementations of PBFT, XFT, FBFT, and ByzCoin; all of them are implemented with the same tools and deployed in the same test-bed with identical restrictions. We choose PBFT as the baseline protocol and representative of a traditional consensus protocol for Byzantine environments. Moreover, XFT serves as the reference for a Byzantine consensus protocol in a synchronous network. Finally, we use FBFT and ByzCoin to compare our results with protocols that improve on top of traditional ones by using collective signing and more efficient communication patterns. In the case of ByzCoin, it uses Schnorr signatures with a tree-based communication protocol. On the other hand, FBFT attempts to improve the inefficient broadcast channels of PBFT by also using BLS threshold signatures. The implementation we used of FBFT uses random gossip as a message propagation protocol.

As expected, XFT and PBFT exhibited poor scalability capabilities owing to the broadcast communication channels. ByzCoin and FBFT significantly improve performance and achieve good network size scalability. Concordia reduces the latency even further by integrating the voting mechanism with the communication protocol, which allows gathering and validating signatures, as blocks propagate through the network in a gossip pattern. Moreover, we allow to securely reach block finality in a single round of voting, further reducing confirmation latency. Overall, our experiments show that Concordia can be up to 4 seconds faster than the closest performing tested protocol.

C. SIGNATURE AGGREGATION

To determine how much time is needed to recover and verify the full BLS signature required to finalize a block, we ran an additional experiment. We measured the time that a node that has received $f + 1$ signatures, needs to compute the final signature. For this purpose, the node verifies each of the signature shares and executes the recovery algorithm. Same as

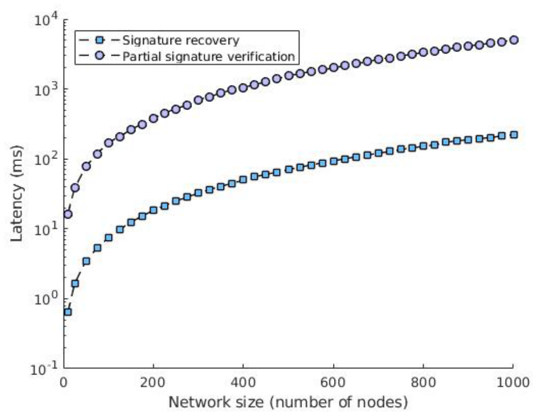


FIGURE 10. Signature scheme benchmarks for different network sizes.

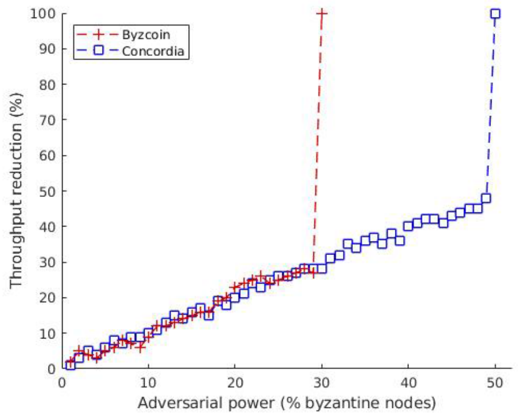


FIGURE 11. Performance decrease comparison under different fractions of adversarial power presence.

in other experiments, we used the BN256 bilinear group [29], and executed the protocol with a network size varying from 10 nodes up to 1000 nodes with a maximum of 500 signatures to aggregate. Fig. 10 shows the time needed to verify the partial signatures and the time needed to execute the recovery algorithm. We decouple signature verification and recovery to progressively verify signatures as they propagate through the network. As soon as a validator gathers enough valid signatures, the recovery algorithm can be executed without needing to verify all the partial signature shares again. As seen in the results, signature verification is considerably more expensive in terms of latency than the recovery. Therefore, we can alleviate this problem by gradually verifying signatures, as they are received.

D. PERFORMANCE UNDER ADVERSARIAL POWER PRESENCE

Owing to the random nature of our block proposer selection method, the probability of selecting a malicious block proposer in a certain round is directly related to the fraction of adversary power in the network. As explained previously, when an incorrect block is proposed, nodes discard it and sign an empty block instead. This means that on those rounds, no transactions are confirmed. Hence, the overall throughput of the system is affected. This problem is not only present in our protocol, but also in other consensus protocols such as ByzCoin, Fig. 11 shows a comparison of the performance reduction due to adversary power presence.

This experiment assumes that all malicious actors try to propose incorrect blocks or not propose any blocks at all. As expected, under this assumption, the performance is directly affected in both cases of our proposal and ByzCoin. Nevertheless, our protocol can keep operating with reduced performance with up to 50% adversary power under our network synchrony assumption.

VI. RELATED WORK

Bitcoin introduces PoW as the consensus mechanism by which miner nodes solve a computational puzzle to win the chance to append a new block. PoW is not capable of supporting instant consensus finality [30]; instead, it relies on ‘‘multi-block confirmations’’ to provide only probabilistic consistency guarantees when the adversary controls less than 50% of the total computing power. As a result, the transaction-confirmation latency of Bitcoin is nearly 1 hour. Moreover, the puzzle computation of the scheme consumes a huge amount of electricity. As an alternative to PoW, proof of stake (PoS) [31] can avoid the computational overhead. Stakeholders in the network have a chance of winning the right to generate a new block, where the winning probability is based on the amount of their stake. Ouroboros [26] is a PoS-based permissionless consensus protocol for cryptocurrencies. The first version of Casper [32] within Ethereum is a hybrid of PoW and PoS, while the ultimate goal of the protocol is to replace the PoW mechanism with a PoS scheme.

Since PBFT protocol found its place in blockchain systems, significant efforts have been made to explore better consensus solutions. One promising direction is to optimize BFT protocols for higher performance [33]. More specifically, a set of protocols have been proposed to accelerate consensus performance by adding a fast execution path under favorable conditions [7]–[9]. For instance, Zyzzyva starts its execution in a quick consensus mode before resorting to the expensive two-phase case. Zyzzyva reduces the all-to-all communication complexity by having the client collect responses from the replicas. This task of signature collection is assigned to designated nodes, called collectors, in SBFT. Additionally, the protocol employs threshold signatures to reduce the communication cost of the BFT variants. However, the protocol still requires two-round communications for signature aggregations. Raft [34] and Tendermint are two popular variants of Paxos and PBFT, respectively, that try to simplify the more complex classic counterparts. Tendermint implements a leader rotation mechanism and scalable transaction propagation using gossip. In contrast, Raft presents an ‘‘understandable consensus algorithm’’ that replicates transactions to all validators, ensuring that they have the same sequence.

Algorand [27] is a consensus scheme based on a Byzantine Agreement (BA) protocol. By combining it with VRFs [35], nodes are chosen to become committee members to participate in the BA protocol and reach consensus on a set of transactions. Dfinity [36] uses the BLS signature scheme as part of its VRF. By using a decentralized random beacon (DRB) protocol, Dfinity participants are allowed to agree on a VRF and jointly produce one new output of the VRF in every round. The agreement of the validator committee can be proved by an aggregated verifiable BLS signature. However, unlike Concordia, since one round of signing cannot be used as a proof of block finalization, Dfinity does not provide full consensus finality, being left vulnerable to fork possibilities.

A new fault model is proposed in XFT which limits the extraordinary power given to an adversary in the original PBFT settings. Its novelty is demonstrated by designing a consensus protocol that embodies the model. However, the protocol operation still relies on the all-to-all communication pattern, which adversely affects the scalability of the protocol together with an inefficient leader election. It is also noteworthy that the protocol introduced in [37] considers a similar network timing model and settings to XFT. The paper presents an efficient Byzantine consensus protocol in a synchronous, authenticated communication network that can tolerate up to $t < n/2$ corrupt nodes. However, it still faces a similar problem, since it also makes use of the all-to-all broadcast communications.

Despite all these efforts, the current technology still falls short of the performance standard for real-world blockchain systems capable of handling tens of thousands of transactions per second. It is widely accepted that this level of TPS can be attained by shard-based consensus protocols such as OmniLedger, RapidChain, Harmony, and Monoxide. OmniLedger uses ByzCoinX, which turns the ByzCoin's multicast model into a two-level tree for robustness. However, the protocol needs to be run by two-level leaders, causing extra communication overhead. Similarly, Rapidchain also relies on a leader-based PBFT variant that tolerates up to $1/2$ malicious nodes under a synchrony assumption. Although Rapidchain presents a more efficient way to propagate large blocks, it still has suboptimal communication complexity that induces higher confirmation latency. Harmony features FBFT as its intra-shard protocol component. As FBFT uses BLS multi-signatures to avoid broadcast channels, it achieves better performance. However, as in ByzCoin and Rapidchain's consensus protocol, Harmony adopts a leader-driven communication pattern. The BLS signature scheme does not directly implement consensus but changes the way for the leader in a BFT protocol to collect and aggregate messages. Thus, Harmony still needs two rounds of signing to achieve consensus. Finally, Monoxide stands out among several research efforts to develop high-performance consensus protocols based on a merged mining scheme. The protocol offers great capacity with substantial throughput and presents a method to deal with cross-shard transactions efficiently. However, Monoxide uses PoW to select block proposers, which entails allowing the possibility of forks and using significant computational power for the mining.

This work has been conducted as a part of our long-term project aiming to develop a high-performance sharding protocol. While designing a novel intra-shard consensus protocol in the first phase, careful considerations were made to achieve our ultimate goal of a high-TPS sharded protocol. It is highly desired that an intra-shard consensus protocol should provide consensus finality to avoid block conflicts, which should help to speed up cross-shard transaction processing. Thus, many of today's shard-based approaches employ a variant of BFT protocols as their intra-shard consensus protocols. Concordia distances itself from the rest in the sense that it is a leaderless and forkless protocol that adopts BLS aggregated signatures as the proof of finality. It is able to avoid block conflicts and expedites transaction processing. Consequently, our intra-shard protocol can provide consensus finality with low latency, which will benefit further cross-shard transaction processing. Already underway as the next phase is our design and integration effort of key components of cross-shard consensus systems such as cross-shard transactions, shard reorganization, incentive mechanisms, and messaging traffic optimization.

VII. CONCLUSION

Concordia streamlines the process of reaching consensus on new blocks, being a highly efficient protocol that achieves great performance while ensuring security. We proposed a new block proposer selection method that inherently prevents the formation of forks. Moreover, by combining the use of a gossip-like communication protocol with a threshold BLS signature scheme, consensus participants can verify and vote on the validity of new blocks in a secure and scalable manner. All components of Concordia are coherently and seamlessly integrated with one another, making it a remarkably efficient consensus protocol for high-throughput blockchain systems.

We implemented a prototype of the proposed protocol with all of its major components in place, to evaluate both its effectiveness and performance. Our evaluation study shows that Concordia has better scaling capabilities in terms of the number of nodes as well as processed transactions, being able to process 2000 TPS with a confirmation latency of approximately 10 seconds in networks of hundreds of nodes. Owing to its security, scalability, and throughput capabilities with remarkably low confirmation latency, Concordia makes for an excellent candidate for an intra-shard consensus protocol for sharded blockchain systems.

ACKNOWLEDGMENT

(Carlos Santiago and Shuyang Ren contributed equally to this work.)

REFERENCES

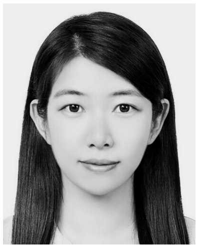
[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," White Paper, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] V. Buterin, "A next-generation smart contract and decentralized application platform," White Paper, 2013. [Online]. Available: https://ethereum.org/en/whitepaper/

[3] D. Schwartz, N. Youngs, and A. Britto, "The ripple protocol consensus algorithm," White Paper, 2014. [Online]. Available: https://ripple.com/files/ripple_consensus_whitepaper.pdf

[4] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with IoT. Challenges and opportunities," *Future Gener. Comput. Syst.*, vol. 88, pp. 173–190, 2018, doi: 10.1016/j.future.2018.05.046.
[5] C. Cachin and M. Vukolić, "Blockchain consensus protocols in the wild," in *Proc. Int. Symp. Distrib. Comput. (DSIC)*, 2017, pp. 1–16.
[6] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 7, pp. 1366–1385, Jul. 2018.
[7] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," *ACM Trans. Comput. Syst.*, vol. 27, no. 4, pp. 1–39, 2010.
[8] P.-L. Aublin, R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić, "The next 700 BFT protocols," *ACM Trans. Comput. Syst.*, vol. 32, no. 4, pp. 1–45, Jan. 2015.
[9] G. Golan Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "SBFT: A scalable and decentralized trust infrastructure," in *Proc. 49th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2019, pp. 568–580.
[10] J. Cao, F. Ellen, L. Rodrigues, and B. Ferreira, "Correctness of tendermint-core blockchains," in *Proc. 22nd ACM Conf. Princ. Distrib. Syst.*, 2018, pp. 1–16.
[11] G. Wang, Z. J. Shi, M. Nixon, and S. Han, "SoK: Sharding on blockchain," in *Proc. 1st ACM Conf. Adv. Financial Technol.*, 2019, pp. 41–61.
[12] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 17–30.
[13] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 583–598.
[14] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, May 2018, pp. 931–948.
[15] H. Team, "Harmony technical white paper," White Paper, 2019. [Online]. Available: https://harmony.one/whitepaper.pdf
[16] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *Proc. 16th USENIX Symp. Netw. Syst. Design Implement.*, Feb. 2019, pp. 95–112.
[17] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme," in *Proc. 6th Int. Workshop Theory Pract. Public Key Cryptogr.*, 2003, pp. 31–46.
[18] L. Lamport, "Paxos made simple, fast, and Byzantine," in *Proc. 6th Int. Conf. Princ. Distrib. Syst.*, 2002, pp. 7–9.
[19] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. USENIX Symp. Operating Syst. Design Implement.*, pp. 173–186, Feb. 22-25, 1999.
[20] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolić, "XFT: Practical fault tolerance beyond crashes," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement.*, Nov. 2016, pp. 485–500.
[21] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," *J. Cryptol.*, vol. 17, no. 4, pp. 297–319, 2004, doi: 10.1007/s00145-004-0314-9.
[22] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, vol. 1592, Aug. 2010, pp. 295–310.
[23] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," in *Proc. 31st Int. Symp. Distrib. Comput. (DISC)*, vol. 91, Oct. 2017, pp. 1–16.
[24] Z. Team, "The Zilliqa technical white paper," White Paper, 2017. [Online]. Available: https://docs.zilliqa.com/whitepaper.pdf
[25] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. and Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *Proc. 25th USENIX Security Symp.*, Aug. 2016, pp. 279–296.
[26] A. Kiayias, A. Russell, B. David, and R. Oliynyko, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Proc. Annu. Int. Cryptol. Conf.*, 2017, pp. 357–388.
[27] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *Proc. 26th Symp. Operating Syst. Princ.*, Oct. 2017, pp. 51–68.
[28] S. Sen and M. J. Freedman, "Commensal Cuckoo: Secure group partitioning for large-scale services," in *Proc. ACM SIGOPS Oper. Syst. Rev.*, 2012, pp. 33–39.
[29] M. Naehrig, R. Niederhagen, and P. Schwabe, "New software speed records for cryptographic pairings," in *Proc. 1st Int. Conf. Cryptol. Inf. Secur. Prog. Cryptol.*, Latin America, Puebla, Mexico, vol. 6212, May 1999, pp. 109–123.
[30] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication," in *Proc. Int. Workshop Open Problems Netw. Secur.*, 2015, pp. 112–125.
[31] S. King and S. Nadal. *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*. Accessed: Aug. 19, 2020. [Online]. Available: https://decred.org/research/king2012.pdf
[32] V. Buterin and V. Griffith, "Casper the friendly finality gadget," 2017, *arXiv:1710.09437*. [Online]. Available: https://arxiv.org/abs/1710.09437
[33] I. Abraham and D. Malkhi, "The blockchain consensus layer and BFT," *Bull. EATCS*, vol. 123, pp. 1–22, Oct. 2017.
[34] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Annu. Tech. Conf.*, 2014, pp. 305–319.
[35] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *Proc. 40th Annu. Symp. Found. Comput. Sci.*, Oct. 1999, pp. 120–130.
[36] T. Hanke, M. Movahedi, and D. Williams, "DFINITY technology overview series, consensus system," 2018, *arXiv:1805.04548*. [Online]. Available: http://arxiv.org/abs/1805.04548
[37] I. Abraham, S. Devadas, D. Dolev, K. Nayak, and L. Ren, "Efficient synchronous Byzantine consensus," 2017, *arXiv:1704.02397*. [Online]. Available: http://arxiv.org/abs/1704.02397

CARLOS SANTIAGO received the B.S. degree in network engineering from the Polytechnic University of Catalonia, Barcelona, Spain, in 2018. He is currently a Graduate Student with Hanyang University, Seoul, South Korea. His research interests include blockchain consensus protocols and distributed systems.



SHUYANG REN received the B.S. degree in information and communication engineering from the North University of China, China, in 2017. She is currently pursuing the Ph.D. degree with Hanyang University, Seoul, South Korea. Her research interests include blockchain consensus protocols and distributed systems.

CHOONHWA LEE received the B.S. and M.S. degrees in computer engineering from Seoul National University, South Korea, in 1990 and 1992, respectively, and the Ph.D. degree in computer engineering from the University of Florida, Gainesville, FL, USA, in 2003. He is currently a Professor with the Department of Computer Science, Hanyang University, Seoul, South Korea. His research interests include cloud computing, peer-to-peer and mobile networking and computing, and distributed computing technology.

MINSOO RYU received the Ph.D. degree from the School of Electrical Engineering and Computer Engineering, Seoul National University, in 2002. He is currently a Professor with the Department of Computer Science and Engineering, Hanyang University, South Korea. His research interests include operating systems, real-time systems, and distributed computing.