Occam: A Secure and Adaptive Scaling Scheme for Permissionless Blockchain

Jie Xu*, Yingying Cheng†, Cong Wang‡, and Xiaohua Jia§
* Department of Computer Science, City University of Hong Kong, Hong Kong
† Huawei Technologies, Hong Kong
* jiexu49-c@my.cityu.edu.hk, † cheng.yingying@huawei.com, ‡ congwang@cityu.edu.hk, § csjia@cityu.edu.hk

2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS) | 978-1-6654-4513-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICDCS51616.2021.00065

***Abstract*—Blockchain scalability is one of the most desired properties for permissionless blockchain. Many recent blockchain protocols have focused on increasing the transaction throughput. However, existing protocols cannot dynamically scale the throughput to meet transaction demand. In this paper, we propose Occam, a secure and adaptive scaling scheme. Occam adaptively changes the transaction throughput by expanding and shrinking according to the transaction demand in the network. We introduce a dynamic adjustment mechanism of mining difficulty and a mining power load balancing mechanism to resist various attacks. Furthermore, we implement Occam on Amazon EC2 cluster with 1000 full nodes. Experimental results show that Occam can greatly increase the throughput of the blockchain and the mining power utilization.**

***Index Terms*—Distributed Consensus, Scalability, Directed Acyclic Graph (DAG), Proof of Work (PoW)**

I. Introduction

Blockchain protocols enable mutually mistrusting entities to agree on a distributed ledger without a central authority. It is expected to be applied to many scenarios to achieve a transparent, trust-free and decentralized system [1–3]. Despite these advantages, the permissionless nature and strong security guarantees of distributed consensus protocols hinder the scalability and confirmation latency. Bitcoin, a representative application of blockchain, can only process seven transactions per second on average, which is far fewer than the processing rate of centralized payment systems such as Visa. Bitcoin uses Nakamoto consensus protocol [4] that organizes transactions into a sequential list of blocks. Miners solve Proof of Work (PoW) puzzles to compete for the opportunity of appending a new block to the end of the longest chain. Although with great simplicity, Nakamoto consensus protocol has led to a huge barrier towards scalability.

There are several good designs [5–9] that have been proposed to increase the scalability of blockchain. Some of them [5, 6] adopt sharding technology, using Byzantine Fault Tolerance (BFT) protocol to achieve consensus within each shard. But, the communication overhead between shards is still very high. Some other studies [8–10] aims to improve the throughput of the blockchain system by changing the blockchain structure into parallel chains or Directed Acyclic Graph (DAG). One of the interesting work among them is OHIE [8]. OHIE is composed of a fixed number of parallel instances of Nakamoto consensus protocol, and each instance follows the longest chain rule. The structure of parallel chains increases the transaction throughput significantly, but fails to provide the dynamic transaction throughput, because the number of instances and mining difficulty are fixed. Conflux [9] applies the Greedy Heaviest Observed Subtree (GHOST) rule to optimistically process concurrent blocks. The purpose of Conflux is to keep all successfully mined blocks on the blockchain, not for responding to a surge of transaction demand. Although it can scale up the throughput and achieve a high mining power utilization, it is difficult to scale down. Conflux ignores the actual transaction demand and does not discard any fork, which incurs unnecessary communication and storage overhead.

It is a challenging issue to dynamic scale up and down transaction throughput while achieving consensus in an open and insecure environment. First, local decisions of scaling up and down in a distributed setting can lead to inconsistent views among participating nodes. According to their local views of the blockchain, some nodes consider that the blockchain needs to scale up, while others consider it is unnecessary. Thus, a global scheme that can make all nodes reach consensus on dynamic scaling needs to be developed. Second, the existing blockchain structure such as single-chain or parallel-chains, is not suitable to scale up or down transaction rate dynamically. The shortest block interval of the blockchain is limited by the network propagation delay, otherwise the blockchain forks and becomes insecure. The blockchain based on single-chain or parallel-chains structure cannot meet the unpredictable dynamic transaction demand due to the lack of adaptability. There is a need to design a dynamic blockchain structure that can expand or shrink according to the transaction demand. Third, concurrent blocks in a dynamic blockchain split the mining power of honest nodes in the system, making the system vulnerable to security attacks. The mining difficulty should be adjusted according as the dynamic expansion and shrink of the blockchain.

In this paper, we present Occam[1], an adaptive scaling scheme for blockchain to meet network dynamic transaction demand with fast transaction confirmation while preserving the safety and liveness. In order to address the first challenge of inconsistent views of blockchain, a global threshold for dynamic scaling is proposed based on the state of the pre-

[1]The word 'Occam' is from 'Occam's razor', which refers to the principle that entities should not be multiplied without necessity.

2575-8411/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDCS51616.2021.00065

vious block that has reached consensus. Since the number of transactions contained in the previous block indicates the transaction demand, when several consecutive blocks are close to the maximum block size, it indicates that the transaction demand is high, and Occam scales up. Similarly, when the blocks are almost empty, Occam scales down. Moreover, all miners follow the *'Deepest Path Rule'*, which means they always choose the view of Occam with the greatest depth to continue to extend. Therefore, all participating nodes can reach consensus on triggering dynamic scaling according to the state of the previous block of the blockchain with the deepest path.

We address the second challenge by allowing blocks to be generated in parallel instead of reducing the block generation interval. Occam is no longer a single-chain structure but takes an exponential expansion and shrink DAG structure. When the transaction demand is high, the mining difficulty decrease and the blockchain expands exponentially; when the transaction demand is low, the mining difficulty increase and the blockchain shrinks exponentially.

For the third challenge, Occam dynamically adjusts the mining difficulty to ensure that the blockchain can continue to extend correctly. At the same time, we also propose a load balancing mechanism on mining power. New blocks are randomly appended to the previous blocks, so that adversaries cannot concentrate their mining power on one concurrent chain to launch attacks. This mechanism can effectively allocate the mining power of the whole system to resist security attacks.

Compared with the existing scaling schemes, Occam has the following advantages:

1) Occam can adaptively scale up and scale down according to the number of transactions in the network. Occam can reduce storage and communication overhead of participating nodes.
2) Occam can dynamically adjust mining difficulty as the blockchain expands and shrinks, which minimize the waste of PoW computing power while ensuring the system security. It can also reduce the confirmation delay, compared with a fix mining difficulty.
3) Occam achieves load balancing of mining power of the whole system, which ensures the safety of the system. The adversaries cannot concentrate their mining power to launch security attacks on a single chain.

The remaining part of the paper is organized as follows. In section II, we introduce the system model and problem statement. Then, in section III, we describe our proposal in detail. Section IV, V and VI analysis the security and performance of Occam. Section VII presents the related work. Finally, section VIII concludes the paper.

II. System Model and Problem Statement

A. System Model

Occam is a fully permissionless blockchain. Any node can participate in the blockchain system. A new participating node can obtain the addresses of other nodes in the network by requesting the existing nodes. In this way, a node can establish connections with other peer nodes. Less than half of the participating nodes in Occam may be controlled by adversaries, while the rest are honest.

Local DAG State. The blockchain in Occam is in a DAG structure, but it could be degenerated into a single chain when transaction demand is low. A new block is linked to previous blocks in the DAG sequentially. There is only one genesis block when initializing the blockchain system. When a new node participates the network, it first obtains the history of the blockchain from other nodes, and stores the history locally. Then, it listens for new blocks being broadcast to the network and verifies them. New valid blocks are added to its local DAG. Blocks of local DAG state are divided into epochs according to the distance from the block to the genesis block. If two blocks are at the same distance from the genesis block, they are in the same epoch. Generally, suppose the distances of blocks B_1 and B_2 to genesis block B^0 are d_1 and d_2 respectively. If $d_1 = d_2$, then B_1 and B_2 are in the same epoch. If $d_1 < d_2$, the epoch of B_1 is before the epoch of B_2; otherwise, B_2 is before B_1.

Transaction Pool. A participating node maintains a temporary set of unconfirmed transactions called the transaction pool, which contains transactions that have been received by their nodes but have not yet been confirmed and added to the blockchain. When a node receives a transaction, it first verifies the transaction. If it is valid and does not exist in the pool, it is added to the pool and broadcast to its neighboring nodes; otherwise the transaction is discarded. When the transaction pool reaches the maximum size, some transactions are discarded based on transaction fees or other criteria. Transactions that have been recorded in the blockchain are also removed away from the transaction pool.

B. Problem Statement

The problem of our concern is to design a secure and adaptive scaling scheme, so that Occam can expand to generate more blocks for recording transactions when there are many transactions in the network, and shrink into a single blockchain when there are few transactions. All honest nodes have the same view of the confirmed block sequence with overwhelming probability. Besides, any transaction generated by an honest node will be recorded on the blockchain after a certain period of time.

III. Design of Secure and Adaptive Scaling Scheme

A. Overview of Occam

Blocks are divided into epochs according to the distance from the block to the genesis block in Occam. The number of valid blocks in the next epoch can be adaptively changed based on the state of the blocks in the previous epochs. If the blocks in the previous consecutive epochs are full, the number of valid blocks in the next epoch doubles, until the maximum number of concurrent blocks is reached. If the blocks in the previous consecutive epochs are empty, the number of valid blocks in the next epoch halves, until there is only one single

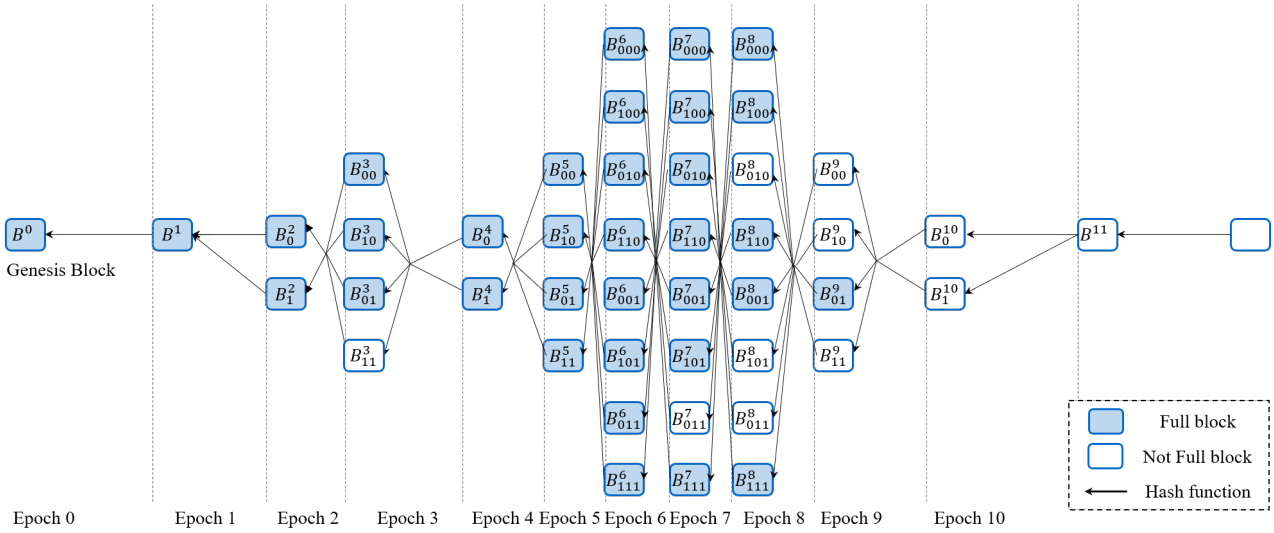


Fig. 1. Overview of Occam adaptive scaling

chain. Figure 1 is an example of Occam which illustrates this dynamic scaling scheme.

Initially, there is only one genesis block, and the blockchain grows one valid block per epoch. If the block is not full, it remains a single blockchain without concurrent blocks. When there are several consecutive blocks that are full and it reaches the threshold to trigger an expansion, then the block is appended by two concurrent blocks in the next epoch. Thus, the number of valid blocks doubles in the next epoch. If the blocks in subsequent consecutive epochs continue to be full and the threshold for expansion has been reached, the number of concurrent blocks in the next epoch doubles again, until it reaches the maximum number of concurrent blocks.

If there are blocks in consecutive epochs that are empty and it reaches the threshold to trigger a shrink, then each block in the next epoch is appended to two concurrent blocks in the previous epoch. Thus, the number of valid blocks is halved of the previous epoch. When the shrink continues, the blockchain eventually degenerates into a single blockchain.

In the following subsection, we explain the adaptive scaling scheme of Occam in details.

B. Adaptive Expansion of Blockchain

Occam uses PoW for mining. There may be multiple concurrent valid blocks in an epoch in Occam. Miners use the root of the merkle tree of all the concurrent blocks in the previous epoch as an input of PoW. The merkle tree is composed of hashes of 2^k blocks in the previous epoch, which are sequenced according to the last k bits of their hashed values. Specifically, let B_i^e be the i^{th} block in e epoch, $i \in [1, 2^k]$. In $e+1$ epoch, miners first calculate the merkle tree composed of $h(B_1^e), h(B_2^e), ..., h(B_{2^k}^e)$, which have different the last k bits. The root of the merkle tree R^e is used as an input to the hash of PoW puzzle in $e+1$ epoch. R^e is also included in all block headers in $e+1$ epoch.

The total mining power for mining in the entire system remains more or less constant even with the increase or decrease of the concurrent blocks. In order to keep the same level of difficulty for mining a block relative to the total available mining power mining this block, the mining difficulty needs to be adjusted each time when the blockchain expands or shrinks. Each time before generating a new block, miners need to determine the number of concurrent blocks in the next epoch.

a) Threshold for Expansion. The threshold for triggering Occam expansion is that the blocks in the previous W_e consecutive epochs are full. In this paper, W_e is set to 1 for simplicity and to show a timely adaptive scaling up. That is, miners determine the number of blocks in the next epoch based on the state of the blocks in the previous epoch.

Whether a block is full is defined by the ratio of the actual block size to the maximum block size. The threshold for full block state is α_u ($0.5 < \alpha_u \leq 1$). The maximum size of a single block is C (for example, Bitcoin is 1M). If the size of a block is greater than $\alpha_u C$, the state of the block is defined as full. When there are 2^k concurrent blocks in an epoch, we use the average block size to measure if the blocks are full or not. If the average size of the blocks is greater than $\alpha_u C$, the blocks in this epoch are regarded as full.

b) Dynamic Adjustment of Mining Difficulty for Expansion. Previous research [8, 11, 12] has pointed out that given a fixed block size C and block propagation delay δ, for any given constant $f < \frac{1}{2}$, there is a constant c, so that the block interval is at least $c \cdot \delta$, and Nakamoto consensus protocol is thus safe. The block interval $c \cdot \delta$ is adjusted by the mining difficulty p. Mining difficulty p means the probability of performing a hash operation to generate a required $nonce$ for PoW puzzle is p, and requires that $hash$ for PoW puzzle to have $log_2 \frac{1}{p}$ leading zeros. Furthermore, considering that there is only one

valid block in each epoch in Nakamoto consensus protocol, the probability that a new block successfully mined in the new epoch is appended to the only valid block in the previous epoch is 1. Therefore, the probability of performing a hash operation to successfully generate a block to be appended after the single blockchain is shown as Eq.(1).

$$p \cdot 1 = p \tag{1}$$

If there are 2^k valid blocks in the previous epoch and the miner determines that there should be 2^{k+1} concurrent blocks in the next epoch, the mining difficulty in the next epoch should become $2^k p$. This means that the probability of a hash operation to successfully get the solution for puzzle is $2^k p$.

The mining power load balancing mechanism is implemented to prevent the adversaries from concentrating their mining power to attack a single chain of the blockchain. The new block successfully mined is randomly appended to one of the 2^k concurrent blocks in the previous epoch. The specific process of appending new blocks to the DAG is in the section III-E. The probability that a successfully mined block is appended to a given block in the previous epoch is $\frac{1}{2^k}$.

As a result, when the blockchain expands, the probability of performing a hash operation, which can successfully generate a block to be appended to a given block, is shown as Eq.(2). The probability that a hash operation successfully generates a block to be appended to a given block is the same as the probability of Nakamoto consensus.

$$2^k p \cdot \frac{1}{2^k} = p \tag{2}$$

C. Adaptive Shrink of Blockchain

a) Threshold for shrink. The threshold for Occam shrinking is that the blocks in the previous consecutive W_s epochs are empty. In this paper, W_s is set to 1. Miners determine the number of blocks in the next epoch based on the state of the blocks in the previous epoch.

The definition of an empty block is also determined by the ratio of the actual block size to the maximum block size. The threshold for the block status to be empty is α_d ($0 \leq \alpha_d \leq 0.5$). When there are 2^k concurrent blocks in an epoch, we use the average block size to measure if the blocks are empty. If the average size of the blocks is less than $\alpha_d C$, the blocks in this epoch are regarded as empty.

b) Dynamic Adjustment of Mining Difficulty for Shrink. If there are 2^k valid blocks in the previous epoch while 2^{k-1} valid blocks in the next epoch, then the mining difficulty of the PoW puzzle in the next epoch is $2^{k-1}p$. This means that the probability of a hash operation to successfully get the solution for puzzle is $2^{k-1}p$. According to the mining power load balancing mechanism, each new block is randomly appended to two of 2^k blocks in the previous epoch. The probability that a new block is appended to a given previous block is $\frac{1}{2^{k-1}}$. Therefore, as shown in the following Eq.(3), the probability that a hash operation successfully generates a new block to be appended to a given previous block is still p.

$$2^{k-1} p \cdot \frac{1}{2^{k-1}} = p \tag{3}$$

D. Parallel Growth of Blockchain

The parallel growth of Occam occurs in two cases: 1) the state of the blocks is stable in the previous epoch; 2) the blockchain has reached the maximum number of concurrent blocks and blocks are continue to be full.

If there are 2^k concurrent blocks in the epoch and the size of the blocks is less than $2^k \alpha_u C$ but greater than $2^k \alpha_d C$, then the state of each block in the epoch is stable. The number of blocks in the next epoch remains the same as the number of blocks in the previous epoch. Occam grows in parallel.

The blockchain cannot expand infinitely, because of the limitation of network bandwidth and network propagation delay. The maximum number of concurrent blocks is set to $2^{k_{max}}$. After reaching the $2^{k_{max}}$ concurrent blocks, if the blocks are still full, then Occam grows in parallel. Note that because k_{max} is determined according to the network bandwidth, it can be increased to upgrade the system when the network bandwidth increased a lot.

When Occam grows in parallel with 2^k concurrent blocks, the mining difficulty of the PoW puzzle is $2^k p$. The probability that a hash operation successfully solves the PoW puzzle is $2^k p$. According to the mining power load balancing mechanism, each new block, each new block is randomly appended to one of the 2^k concurrent blocks in the previous epoch, so the probability that a new block is appended to a given block in the previous epoch is $\frac{1}{2^k}$. When the blockchain grows in parallel, the probability of performing a hash operation, which can successfully generate a block to be appended to a given block, shown as Eq.(4).

$$2^k p \cdot \frac{1}{2^k} = p \tag{4}$$

E. Appending New Blocks to Blockchain

a) Unique Ordering of Concurrent Blocks in an Epoch. When there are multiple blocks in an epoch, they are concurrent blocks. To ensure all nodes eventually agree on the same set of blocks in each epoch, Occam uses the last k bits of the hash values of the blocks to determine the order of blocks in each epoch, supposing there are 2^k blocks in this epoch. To avoid the situation where two concurrent blocks whose the last k bits of their hashes are the same, Occam defines that, for an epoch with 2^k concurrent blocks, the last k bits of the hash values of the 2^k blocks should be different. For example, if there are 2^2 blocks in an epoch, the last 2 bits of the hashes of the 4 blocks should be 00, 10, 01, and 11, respectively. If the miner receives more than one block with the same last k bits of hashes of the block in the same epoch, the miner chooses one of the blocks according to receiving time. The miner cannot mine blocks in the next epoch, until it receives all the 2^k blocks with the different last k bits in the current epoch. The 2^k blocks in an epoch are ordered by the last k bits of their hash values. As shown in Figure 1, blocks in epoch 5 are ordered by their last 2 bits, and in epoch 6 they are ordered by the last 3 bits.

b) Mining Power Load Balancing. The mining power load balancing mechanism effectively allocates the mining power

of the whole network to each path of the blockchain, which can resist fraud attacks, liveness attacks and so on.

In Bitcoin, there is only one path, the longest chain. The path is formed by containing the hash of the previous block in the header of the next block, which acts as a pointer to the parent block. In Occam, blocks are linked as a DAG structure with multiple paths, as shown in Figure 2. The parent of a new block is determined by the last few bits of their hash values. When no expansion or shrink, supposing there are 2^k blocks in the last epoch, the parent of the new block must have the same last k bits as the new block. When at expansion, the number of blocks in the next epoch is 2^{k+1}. The block in the last epoch that have the same last k bits as the new generated block is its parent. When at shrink, a block has two parent blocks. If there are 2^k blocks in the next epoch, the two blocks in the last epoch that have the same last k bits as the new generated block are its parents.

As the miner cannot know which block is the parent of the new block until it has solved the PoW puzzle and calculated the $hash$ of the new block, new blocks are randomly appended to blockchain. Miners cannot pre-determine which path to follow. Thus, the mining power is randomly distributed among all the concurrent paths and no miner is able to concentrate its mining power on a single path.

When other nodes receive a new block, the nodes learn the parent node of the new block by comparing the hashes of all blocks in the previous epoch contained in the new block with the hash of the new block. If there are 2^k blocks in e epoch and 2^{k+1} blocks in the next epoch, each of these 2^{k+1} new blocks is appended to the block with the same last k bits of the hash of the block in the previous e epoch. Therefore, two concurrent blocks with the same last k bits of the hash of the block can be appended to the same block in the e epoch. For example, in Figure 2, both B^3_{00} and B^3_{10} are appended to B^2_0, because the last 1 bit of $h(B^3_{00})$ and $h(B^3_{10})$ are both 0, which is the same as the last 1 bit of $h(B^2_0)$. However, B^3_{01} and B^3_{11} cannot be appended to B^2_0, because the last 1 bit of $h(B^3_{01})$ and $h(B^3_{11})$ are 1. When Occam grows in parallel, the new block is appended to the block in the previous epoch with the same last k bits of the hash of the block.

If there are 2^k blocks in e epoch and 2^{k-1} blocks in $e+1$ epoch, each block in $e+1$ epoch is appended to the previous block with the same last $k-1$ bits of the hash of the block in the previous e epoch. This means that each block in $e+1$ epoch is appended to two blocks in e epoch, because the last $k-1$ bits of the hash of these blocks are the same. For example, B^4_0 is appended to B^3_{00} and B^3_{10}, because the last bit of $h(B^4_0)$ is 0, which is the same as the last 1 bit of $h(B^3_{00})$ and $h(B^3_{10})$.

c) The Deepest Path Rule. Blockchain inevitably has forks because of network propagation delay. All miners always follow the authoritative Occam view to continue mining according to the *'Deepest Path Rule'*.

Due to network delays, new blocks cannot be received by all nodes immediately and simultaneously. Thus, it is very likely that a puzzle has been solved, but other nodes have not received this message, so they continue to solve the puzzle

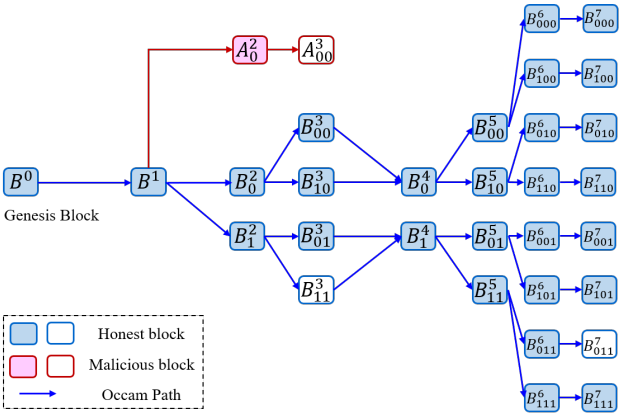


Fig. 2. The deepest path rule

and might generate another new block. Nakamoto recommends that miners adopt the longest chain rule [4] to deal with this problem. That is, only the longest chain is correct and continue to be extended. As a result, the eventual consistency of the blockchain can be achieved. Similarly, in Occam the chain with the greatest depth is authoritative and correct. Honest participating nodes always choose to continue mining on the deepest blockchain. When receiving multiple blocks in the same epoch with the same last k bits of the hash of block, the miner first discards the block that contain invalid transactions. Then, it chooses one of them to be hashed in the merkle tree. Thus, the merkle tree is composed of 2^k blocks with the different last k bits of the hash of block. The root of the merkle tree is used as an input for mining in the next epoch. Note that miners also record other verified blocks that are not chosen in case other miners choose to follow them. If the first new block in the next epoch is received, miners switch to follow the choice of the miner who generated the first new block in the next epoch. Therefore, all blocks in the new epoch contain the same merkle tree of the previous epoch. Then, miners continue to mine blocks on other paths in the next epoch. In general, the last T blocks on each path are unconfirmed blocks, while the previous blocks are confirmed.

F. Resolving Conflicting Transactions

Since the blockchain network is an asynchronous network and miners prefer to choose transactions with higher transaction fees, it is possible that blocks generated in the same epoch contain the same transactions. Although further efforts are needed to optimize this problem, it only affects efficiency but not security and scalability. If multiple blocks in the same epoch contain the same transaction, only the first transactions is processed according to the global transaction order. Subsequent identical transactions are ignored. The miners who generate subsequent blocks containing the same transaction cannot get transaction fees, even if the block is still valid and exists on the blockchain. A possible solution avoiding including transactions in the same epoch to divide the transaction into different groups according to the last few

bits of the identity of transactions. This will be our future work.

There are not only duplicate transactions, but also conflicting transactions in the network. If a block in subsequent epoch contains a transaction that conflicts with the transaction contained in the previous block, nodes discard the later block. If the transactions contained in different blocks in the same epoch conflict, only the block with the relative lower order is kept according to the order of the blocks, and the subsequent blocks are considered invalid and discarded. The order of blocks in Occam first follows the epoch order. Then, the 2^k concurrent blocks in the same epoch are ordered according to the last k bits of the hash of the block. After collecting blocks on all paths in the epoch, and there are no conflicting transactions between these blocks, miners determine the number of the blocks in the next epoch and continue mining.

IV. Security Analysis

A. Security Assumptions

We assume that Occam system is in a static setting composed of fixed number of N participating nodes, but the participating nodes do not know how many nodes participate in the system. As pointed out in the literature [13], a deterministic consensus protocol cannot be reached in a completely asynchronous network. Occam is based on the assumption of a synchronous network. All messages can be received within a known time span. Occam proceeds in rounds, and all participating nodes in the network can synchronize within Δ rounds. The hash function of PoW puzzle is modeled as a random oracle in Occam. In each round, each node receives some message from network and queries the random oracle. All honest nodes only adopt the 'deepest tree rule' as the tie-breaker.

B. Threat Model

The adversary $\mathcal{A}$ can delay or reorder all messages in the network, as long as the messages can be eventually broadcast to other nodes. The adversary $\mathcal{A}$ is able to delay the message up to Δ rounds. That is, if any node to send a message m at round r, all participating nodes in the network must receive the message on or before $r + \Delta$. The adversary can adopt various attack strategies, such as broadcast different information to different honest nodes, or can broadcast multiple times in each round. However, the adversary cannot modify the content of the message or prevent message broadcasting.

Any node can access oracle $H()$ with the mining difficulty p, which can be regarded as the probability of a single random oracle query to successfully mining. The mining capacity of each node is equal. The adversary corrupted at most f fraction of the total N participating nodes, and f is less than $\frac{1}{2}$. Therefore, in each round, adversary is able to query $f \cdot N$ times random oracle. The adversary $\mathcal{A}$ can corrupt and control targeted nodes in the network anytime, but adversary is computationally bounded, which means they have no reverse cryptography capabilities. $\mathcal{A}$ cannot crack the one-way hash function and obtain the user's private key from the user's public key.

C. Safety Analysis

Theorem 1: *All honest nodes have the same view after expansion.*

Proof: If the states of the 2^{k-1} blocks in $e-1$ epoch are full, Occam starts to expand. Even if there is more than one block for a path in e epoch, that is, even if there are forks in epoch e, once the first block B^{e+1} in $e+1$ epoch is generated, honest nodes follow the choice in block B^{e+1} because B^{e+1} is the deepest at this time. That is, honest nodes switch to $e+1$ epoch mining to generate blocks containing the same blocks of the e epoch as contained in B^{e+1}. Therefore, all honest nodes still have the same view after expansion.

Theorem 2: *All honest nodes have the same view after shrink.*

Proof: If the states of 2^{k-1} blocks in $e-1$ epoch are empty, Occam starts to shrink. Even if there is more than one block for a path in e epoch, that is, even if there are forks in epoch e, once the first block B^{e+1} in $e+1$ epoch is generated, honest nodes follow the choice in B^{e+1} block because B^{e+1} is the deepest at this time. That is, honest nodes switch to $e+1$ epoch mining to generate blocks containing the same blocks of the e epoch as contained in B^{e+1}. Therefore, all honest nodes still have the same view after shrink.

Theorem 3: All nodes eventually maintain the same view of blockchain in Occam.

Proof: If there are 2^k blocks in e epoch, due to propagation delay, some participating nodes receive 2^k blocks are full and believe the blockchain should expand in $e+1$ epoch, and other nodes receive 2^k blocks are empty and they believe the blockchain should shrink in epoch $e+1$. Because if the blockchain expands in $e+1$ epoch, the mining difficulty is $2^k p$, which is less than the mining difficulty $2^{k-1}p$ when the blockchain shrink in $e+1$ epoch. The group of miners who choose to expand in $e+1$ epoch will eventually generate a blockchain with deeper depth. According to the deepest path rule, the group of miners who chose to shrink in $e+1$ epoch also switch to continue mine the blockchain generated by the miners who chose to expand in $e+1$ epoch. In addition, once a block is added to the blockchain, the epoch to which it belongs is determined, and the hash of the block is determined, so the order of the block in the epoch is determined, and the order of the transactions contained in the block is also unique and determined based on the global transaction order algorithm.

Combining **Theorem 1** and **Theorem 2**, we can get all the nodes eventually maintain the same Occam view regardless expansion or shrink, even if there is a difference from time to time. Therefore, similar to Bitcoin, Occam achieves eventually consistency. That is, with overwhelming probability, at any point, the state of the local DAG of all honest nodes can differ only in the last several blocks.

D. Liveness Analysis

With the dynamic adjustment of mining difficulty and load balancing of mining power, regardless expansion or shrink, the probability of performing a random oracle query to successfully generate a block to after a certain block is p, which is the same as Nakamoto consensus mechanism. As long as the compute power controlled by the adversary is less than half, the probability of an adversary generating an expanded blockchain to catch up with the existing blockchain maintained by an honest node becomes negligible as the epoch increases. Therefore, Occam has the same liveness property as the Nakamoto consensus protocol. Once the depth of the block exceeds T in the local view of an honest node, liveness attack cannot succeed.

V. Implementation

A. Overview

Each node in our design maintains a local blockchain that is a tree of blocks. All blocks connect to the genesis block through paths. Each node in our protocol updates this tree by a mining thread and a receiving thread. The mining thread keeps solving the PoW puzzle, and then broadcasts the generated block. The receiving thread processes the blocks received from other peer nodes. When the mining thread calls Algorithm 1 to generate a new block, the node keeps receiving blocks but does not process the received blocks. Until the mining thread ends, the receiving thread calls Algorithm 2 to synchronize the state of the blockchain.

B. Adaptive Block Generation

The mining thread generates blocks by following the procedure in Algorithm 1. The node first judges whether the blockchain is expanding or shrinking or growing in parallel from the state of the blocks in the previous epoch. If the state of the blocks is stable, the mining difficulty is the same as that in the previous epoch, which is still $2^k p_0$, as shown in line 6 to 8. If it expands or shrinks, the node needs to call the expansion function or shrink function to adjust the difficulty and continue mining.

Algorithm 1 Adaptive Block Generation

Input: Set of 2^k blocks $\mathcal{B}^e = \left[B_1^e, B_2^e, ..., B_{2^k}^e\right]$ in epoch e, initial mining difficulty p_0, and set of transactions $\mathcal{T}^{e+1} = \left[tx_1^{e+1}, tx_2^{e+1}, ..., tx_n^{e+1}\right]$ for epoch $e+1$
Output: New Blocks in epoch $e+1$

1: **if** Blocks in $\mathcal{B}^e$ are full **then**
2: EXPAND$(k, \mathcal{B}^e, \mathcal{T}^{e+1})$
3: **else if** Blocks in $\mathcal{B}^e$ are empty **then**
4: SHRINK$(k, \mathcal{B}^e, \mathcal{T}^{e+1})$
5: **else**
6: $R^{e+1} \leftarrow MerkleTree\_root(\mathcal{B}^e)$
7: $p \leftarrow 2^k p_0$
8: PoW $(R^{e+1}, \mathcal{T}^{e+1}, p)$
9: **end if**

Figure 3 is the function of expansion. If the blockchain expands and the number of concurrent blocks does not exceed the maximum number of concurrent blocks, the mining difficulty is adjusted to $2^k p_0$. The number of concurrent blocks in the next epoch is doubled, that is, $k = k + 1$. After reaching the maximum number of concurrent blocks $2^{k_{max}}$, even if the state of the blockchain is expanding, the number of concurrent blocks and the mining difficulty do not change anymore.

1: **function** EXPAND$(k, \mathcal{B}^e, \mathcal{T}^{e+1})$
2: $R^{e+1} \leftarrow MerkleTree\_root(\mathcal{B}^e)$
3: **if** $2^k < 2^{k_{max}}$ **then**
4: $p \leftarrow 2^k * p_0$
5: $k \leftarrow k + 1$
6: $PoW(R^{e+1}, \mathcal{T}^{e+1}, p)$
7: **else**
8: $p \leftarrow 2^{k_{max}} * p_0$
9: $PoW(R^{e+1}, \mathcal{T}^{e+1}, p)$
10: **end if**
11: **end function**

Fig. 3. The function of expansion

Figure 4 is the function of shrink. If the number of concurrent blocks in the previous epoch is greater than 1, the mining difficulty in the next epoch is halved, changing to $2^{k-1} p_0$. The number of concurrent blocks in the next epoch is also halved. Otherwise, the blockchain maintains a single chain for mining and the mining difficulty is p_0.

1: **function** SHRINK$(k, \mathcal{B}^e, \mathcal{T}^{e+1})$
2: $R^{e+1} \leftarrow MerkleTree\_root(\mathcal{B}^e)$
3: **if** $2^k > 1$ **then**
4: $p \leftarrow 2^{k-1} * p_0$
5: $k \leftarrow k - 1$
6: $PoW(R^{e+1}, \mathcal{T}^{e+1}, p)$
7: **else**
8: $PoW(R^{e+1}, \mathcal{T}^{e+1}, p_0)$
9: **end if**
10: **end function**

Fig. 4. The function of shrink

C. Processing Received Blocks

The receiving thread processes each received block according to Algorithm 2. The node first verifies whether the received block and its parents blocks have conflicting transactions. Finally, the node appends the block to the local view according to the last few bits of the hash of the received block.

As it is shown in line 3 in Algorithm 2, a block is considered to be appended if its depth is no less than that of the current local graph. If the block is valid, we further check the mining difficulty of the block. The node first gets the state of the blocks in the previous epoch of the received block. As it is shown in line 7 to 13, there are three cases for appending

a received block to local graph. Different states of blocks in the last epoch have different requirements for the mining difficulty of the received block. If the mining difficulty of the received block is not lower than the required mining difficulty, the received block is appended to the local DAG.

Algorithm 2 Process Received Block

Input: The receiving new block B' and local graph G
Output: Graph G'

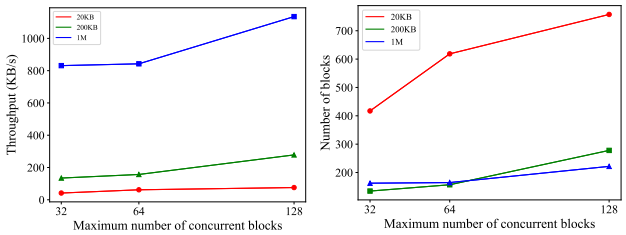
$d_{B'} \leftarrow \text{DISTANCE}(B', B^0)$
if B' and its parent blocks are all valid **then**
  if $d_{B'}$ is no less than the depth of G **then**
    $z_{B'} \leftarrow$ the number of leading zeroes in $hash(B')$
    $\mathcal{B} \leftarrow$ the block set with depth $d_{B'} - 1$
    $2^{\hat{k}} \leftarrow$ the number of blocks in $\mathcal{B}$
    if blocks in $\mathcal{B}$ are full and $z_{B'} \geq log_2 \frac{1}{2^{\hat{k}}p}$ **then**
      APPENDING$(B', G, d_{B'}, \hat{k}+1)$
    else if blocks in $\mathcal{B}$ are empty and $z_{B'} \geq log_2 \frac{1}{2^{\hat{k}-1}p}$ **then**
      APPENDING$(B', G, d_{B'}, \hat{k}-1)$
    else if blocks in $\mathcal{B}$ are stable and $z_{B'} \geq log_2 \frac{1}{2^{\hat{k}}p}$ **then**
      APPENDING$(B', G, d_{B'}, \hat{k})$
    end if
  end if
end if

VI. Experimental Evaluation

The prototype of Occam is implemented in C++. We conduct Occam on Amazon EC2 platform. The Amazon's EC2 virtual machines are m4.2xlarge virtual machines (VMs), each of which has 8 cores and 1000 Mbps bandwidth. We rent 20 virtual machines, which are distributed in 14 different cities around the world. In order to measure the performance of Occam for large-scale users, each virtual machine simulated 50 participating nodes. Therefore, our final experimental results are based on the results of 1000 participating nodes. The maximum bandwidth of each participating node is 20 Mbps. All connections between nodes are peer-to-peer. Each node establishes connections with 8 randomly selected peer nodes. Whenever a node generates a new block, it is first broadcast to its 8 peer nodes, and then broadcast to all peer nodes in the network through the gossip protocol. Initially, we set the expected value of block generation interval for all nodes is 16 seconds. After each expansion, the expected value of the block generation time is halved. Taking into account the block transmission delay, we set the shortest expected value of block generation interval to 1 second.

A. Block Size

Block size is an important parameter that affects blockchain throughput and blockchain safety. We set up three sets of experiments. The three experiments are at 8 Mbps bandwidth, and the expected value of block generation interval for all nodes is 1 second. We change the size of blocks, and then compare throughput and the number of blocks that have reached the consensus.



(a) (b)

Fig. 5. The impact of block size on (a) throughput, and (b) the number of consensus blocks

As shown in Figure 5(a), regardless of the block size, the more concurrent blocks, the greater the throughput of the blockchain. The throughput of a blockchain with the block size of 1 M is much greater than that of a blockchain with the block size of 20 KB.

However, according to the Figure 5(b), the number of blocks that have reached the consensus with the size of 20 KB far exceeds the number of blocks that have reached the consensus with the size of 1 M. Such experimental results are reasonable. The verification delay and transmission delay of smaller blocks are shorter, so the smaller blocks propagate faster in the network.

B. Dynamic Scalability

In order to verify the dynamic throughput of our scheme, we conducted 5 sets of experiments with the bandwidth of 20 Mbps and the block size of 20 KB. We set the maximum number of concurrent blocks to 1, 2, 4, 8, and 16 respectively. The initial mining difficulty of each participating node is the same and the expected value of block generation interval is 16 seconds. The expected value of block generation interval is halved after each expansion. In these five sets of experiments, the state of all blocks in the blockchain are always full. The blockchain runs for 200 seconds and we get the average throughput of 1000 nodes. The results are shown in Figure 6.

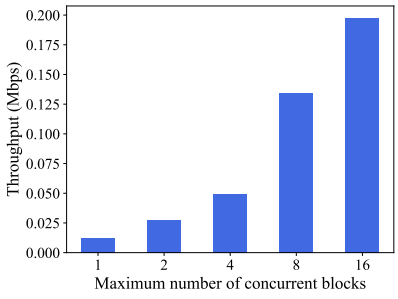


Fig. 6. Dynamic throughput with different number of concurrent blocks

As shown in Figure 6, the throughput of the blockchain increases significantly as the maximum number of concurrent blocks increases. When the blockchain is always a single chain, the throughput of the blockchain is about $0.01\,\mathrm{Mbps}$. When the blockchain grows from a single block to the DAG with 16 concurrent blocks, the throughput reaches up to $0.2\,\mathrm{Mbps}$.

C. Mining Power Utilization

The ratio of the number of confirmed blocks on the blockchain to the total number of generated blocks is used to show the mining power utilization. We compare 4 sets of experiments where the maximum number of concurrent blocks is set to 1, 2, 4, and 8, respectively. We measure the number of confirmed blocks and all blocks generated in the network when the bandwidth is $20\,\mathrm{Mbps}$ and the block size is $20\,\mathrm{KB}$. The experimental results are shown in the figure below.

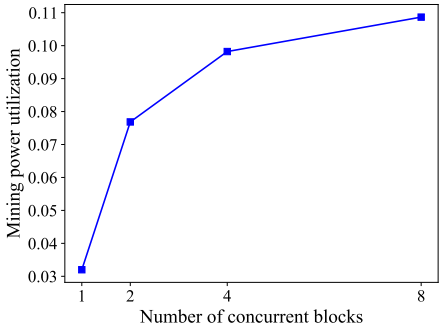


Fig. 7. Mining power utilization

As expected, Figure 7 shows the mining power utilization increases as the number of concurrent blocks increases. The mining power utilization of the blockchain with single chain structure, such as Bitcoin, is very low, only 0.03. The mining power utilization of Occam with exponential expansion and shrink DAG structure is much higher than that of the single chain. When Occam reaches and grows by 8 concurrent blocks, the mining power utilization is up to 0.11.

D. Confirmation Delay

The last T blocks on each path is regarded as unconfirmed block, and the previous blocks are regarded as confirmed blocks. The records on the confirmed blocks has been verified and cannot be modified or deleted. All nodes have a consistent view on the confirmed block. Figure 8 shows the average confirmation delay of Occam with a block size of $20\,\mathrm{KB}$ and 128 parallel chains under different bandwidths and T settings.

Our results show that as T increases, the confirmation delay increases significantly. If the T value is the same, the larger the bandwidth, the shorter the confirmation delay. When $T = 3$, the average confirmation delay of the blockchain under $8\,\mathrm{Mbps}$ is $62.98\,\mathrm{s}$, and the average confirmation delay of the blockchain under $20\,\mathrm{Mbps}$ is $42.58\,\mathrm{s}$.

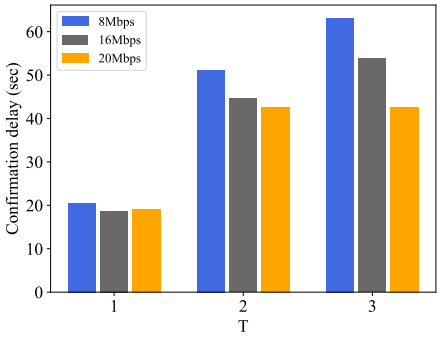


Fig. 8. Confirmation delay

VII. Related Work

Off-chain scaling. Many approaches have been applied to the cryptocurrency blockchain to improve its scalability, such as Segregated Witness (SegWit) [14] and Micropayment Channel [15] or Lightning Network [16] for Bitcoin [4]. Segregated Witness [14] is a technology that has been applied to Bitcoin. It does not increase the block size, but achieves more transactions per block by reducing the amount of information stored per transaction. Specifically, it removes the unlock signature (witness data) in the transaction and extends it to the end as a separate structure. In this way, more transactions can be stored in a block. However, by removing the witness data, blockchain throughput that can be increased is still limited. In Micropayment Channel [15] and Lightning Network [16] , only the funding transaction and settlement transaction are on-chain, all commitment transactions are off-chain. Micropayment Channel and Lightning Network are solutions reduce the number of user interactions with the blockchain to reduce transaction delays, but they cannot increase the throughput of the blockchain.

In addition to the above-mentioned research done on layer 2 (off-chain), there are also some works dedicated to scale blockchain on layer 1 (on chain). Different from the scheme on layer 2, schemes on layer 1 mainly focuses on increasing the throughput of blockchain by reducing communication or computing overhead of the nodes.

Consensus. In order to improve the efficiency of Nakamoto consensus protocol [4], Eyal *et al.*[7] designed a new consensus named Bitcoin-NG, which decouple leader election and transaction serialization. The leader is selected through Proof-of-Work (PoW) and recorded in the key block. Unlike Bitcoin, a leader selected in each epoch can generate multiple microblocks to record packaging transactions until a new leader is chosen. Although the Eyal *et al.* introduced *poison transactions* to invalidate the income of malicious leader to avoid double-spending attacks, Bitcoin-NG still requires a long transaction confirmation latency. Based on Bitcoin-NG, to reduce the confirmation time, Kokoris-Kogias *et al.* [17] introduce Byzcoin that utilizes scalable collective signatures to irreversibly commit transactions. In Byzcoin, there is a fixed-size sliding share window to adaptivelyly form an opening

consensus group to commit PBFT consensus [18], so it also solve the consistency weakness of Bitcoin. In addition, Solida [19] present a reconfigurable Byzantine consensus using PoW for committee elections. The above are some solutions to reduce the consensus time of the block generation interval. The following are a number of initiatives aimed at generating multiple blocks between each block generation interval.

Sharding. Sharding is a promising technique to improve the horizontal scalability. Nodes in the entire blockchain system are divided into different shards. Each shard handles only a small part of all transactions, and consensus can be reached within the shard. As long as there are enough nodes to verify each transaction, the system is still highly secure. As a result, blockchain system can process more transactions in parallel to improve scalability. Elastico[5] is arguably the first literature to propose sharding technique in blockchain. In 2016, Luu *et al.* proposed Elastico [5], which divides the nodes into smaller committees based on the least significant bits of the PoW hash. Each committee handles sharding and runs PBFT to reach consensus. Eventually agreed values of committees are shrinkd and a new block is generated. However, several drawbacks still exist in Elastico, and transactions between shards remains a huge challenge. After that, Kokoris-Kogias *et al.* [20] established OmniLedger on the basis of Elastico. OmniLedger uses an Atomic Commit protocol to ensure transaction atomicity across shards, and introduce a Byzantine consensus protocol to resist DoS attack. OmniLedger is vulnerable to Denial-of-Service (DoS). The cross-shard transaction processing in Omniledger is still insecure. A malicious recipient can block the payer's payment. Furthermore, Rapidchain[21] achieve lower communication overhead than Omniledger, and improve throughput and Byzantine faults resiliency, but it does not achieve isolation. Recently, Dang *et al.* [6] introduces trusted hardware to increased Byzantine fault tolerance and generate unbiased random values for secure shard formation. However, efficiently dividing transactions into different shards and the throughput of inter-shard transactions remain a huge challenge.

DAG-based Structures. For the purpose of improving blockchain scalability, some approaches no longer use a single chain structure but a Directed Acyclic Graph (DAG) to record more transactions. In Nakamoto consensus, concurrent blocks are invalid or even harmful and will eventually be discarded to ensure the longest-path-rule. Nevertheless, Conflux [10] proposes to optimistically processes concurrent blocks without discarding any fork to increase the throughput of the blockchain. Thus, the overall structure of Conflux is no longer a single blockchain but a Tree-Graph structure. Considering that the tree-graph structure is vulnerable to liveness attacks, Conflux designed an adaptive weight mechanism to ensure the security of consensus. In 2020, Yu *et al.* propose OHIE [8], which composes several parallel instances of Nakamoto consensus. Although it seems that OHIE is a parallel chains structure, in fact, OHIE can be considered as a kind of DAG scheme considering that there are hash mappings in blocks of other parallel chains. Due to the $confirm\_bar$ needs to wait for all parallel chains to generate new blocks to move forward, the confirmation delay in OHIE is very long. In addition, there are also some blockchain projects that use DAG structure, such as IOTA [22], and Byteball [23]. They are representative DAG blockchain projects suitable for IoT scenarios. Their consensus protocol takes into account the limited computing and storage capabilities of IoT devices, but it also brings defects such as uncontrollable transaction confirmation delays.

VIII. Conclusion

We presented Occam, a new blockchain scheme that can dynamically scale up and down according to the transaction demand of the network. When the transactions demand is high, it scales up by allowing more concurrent blocks in an epoch; when the transaction demand is low, it scales down by reducing the number of concurrent blocks in an epoch. It preserves safety and liveness by dynamic adjusting mining difficulty and performing load balancing on mining power. Our scheme is implemented and deployed on Amazon EC2 cluster. The experiment results show that our proposed protocol achieves the dynamic throughput and a high utilization of mining power.

Acknowledgment

This work was supported by grant from Research Grants Council of Hong Kong [Project No. CityU 11202419].

References

[1] K. Huang, X. Zhang, Y. Mu, X. Wang, G. Yang, X. Du, F. Rezaeibagha, Q. Xia, and M. Guizani, "Building redactable consortium blockchain for industrial Internet-of-Things," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3670–3679, 2019.

[2] C. Xu, K. Wang, P. Li, S. Guo, J. Luo, B. Ye, and M. Guo, "Making big data open in edges: A resource-efficient blockchain-based approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 870 – 882, 2019.

[3] C. Chen, J. Wu, H. Lin, W. Chen, and Z. Zheng, "A secure and efficient blockchain-based data trading approach for internet of vehicles," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 9, pp. 9110 – 9121, 2019.

[4] S. Nakamoto. (2008) Bitcoin: A peer-to-peer electronic cash system. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[5] L. Luu, V. Narayanan, C. Zheng, K. Baweja, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 17–30.

[6] H. Dang, T. T. A. Dinh, D. Loghin, E. C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 123–140, 2019.

[7] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-NG: A Scalable Blockchain Protocol,"

in *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*. USENIX Association, 2016, pp. 45–59.

[8] H. Yu, I. Nikolić, R. Hou, and P. Saxena, "OHIE: Blockchain scaling made simple," in *Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 90–105.

[9] C. Li, P. Li, W. Xu, F. Long, and A. Yao, "Scaling nakamoto consensus to thousands of transactions per second," *CoRR*, vol. abs/1805.03870, 2018. [Online]. Available: http://arxiv.org/abs/1805.03870

[10] C. Li, P. Li, D. Zhou, Z. Yang, M. Wu, G. Yang, W. Xu, F. Long, and A. C.-C. Yao, "A decentralized blockchain with high throughput and fast confirmation," in *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 515–528.

[11] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 643–673.

[12] L. Kiffer, R. Rajaraman, and A. Shelat, "A better method to analyze blockchain consistency," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 729–744.

[13] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[14] E. Lombrozo, J. Lau, and P. Wuille, "Segregated witness (consensus layer)," *Bitcoin Core Develop. Team, Tech. Rep. BIP*, vol. 141, 2015.

[15] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment

[20] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, channels," *Symposium on Self-Stabilizing Systems*, vol. 9212, pp. 3–18, 2015.

[16] J. Poon and T. Dryja. (2016) The bitcoin lightning network: Scalable off-chain instant payments. [Online]. Available: https://www.bitcoinlightning.com/wp-content/uploads/2018/03/lightning-network-paper.pdf

[17] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing Bitcoin security and performance with strong consistency via collective signing," in *Proceedings of the 25th USENIX Security Symposium*. USENIX Association, 2016, pp. 279–296.

[18] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. The Advanced Computing Systems Association, 1999, pp. 173–186.

[19] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman, "Solida: A blockchain protocol based on reconfigurable Byzantine consensus," *Leibniz International Proceedings in Informatics, LIPIcs*, vol. 95, pp. 1–17, 2018.

and E. Syta, "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding," in *Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 583–598.

[21] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 931–948.

[22] S. Popov. (2018) The Tangle. [Online]. Available: http://www.descryptions.com/Iota.pdf

[23] A. Churyumov. (2018) Byteball: A decentralized system for storage and transfer of value. [Online]. Available: https://byteball.org/Byteball.pdf