Eunomia: A Permissionless Parallel Chain Protocol Based on Logical Clock

Jianyu Niu
School of Engineering, University of British Columbia (Okanagan Campus), Kelowna, Canada
jianyu.niu@ubc.ca

***Abstract*—The emerging parallel chain protocols represent a breakthrough to address the scalability of blockchain. By composing multiple parallel chain instances, the whole systems' throughput can approach the network capacity. How to coordinate different chains' blocks and to construct them into a global ordering is critical to the performance of parallel chain protocol. However, the existed solutions use either the global synchronization clock with the single-chain bottleneck or predefined ordering sequences with distortion of blocks' causality to order blocks. In addition, the prior ordering methods rely on that honest participants faithfully follow the ordering protocol, but remain silent for any denial of ordering (DoR) attack.**

On the other hand, the conflicting transactions included into the global block sequence will make Simple Payment Verification (SPV) difficult. Clients usually need to store a full record of transactions to distinguish the conflictions and tell whether transactions are confirmed. However, the requirement for a full record will greatly hinder blockchains' application, especially for mobile scenarios.

In this technical report , we propose Eunomia, which leverages logical clock and fine-grained UTXO sharding to realize a simple, efficient, secure and permissionless parallel chain protocol. By observing the characteristics of the parallel chain, we find the blocks ordering issue in parallel chain has many similarities with the event ordering in distributed system. Eunomia thus adopts "virtual" logical clock, which is optimized to have the minimum protocol overhead and runs in a distributed way. In addition, Eunomia combines the mining incentive with block ordering, providing incentive compatibility against DoR attack. What's more, the fine-grained UTXO sharding does well solve the conflicting transactions in parallel chain and is shown to be SPV-friendly.

***Index Terms*—Blockchain, Parallel Chain, Bitcoin, Proof-of-Work, UTXO, Logical Clock**

I. Introduction

In 2008, Nakamoto invented the seminal *blockchain* protocol which uses the *Nakamoto Consensus (NC)* to realize a public, immutable and distributed ledger, Bitcoin [1]. In NC, participates, called miners are allowed to generate a new block — a collection of transactions by solving computational puzzles, known as *Proof-of-Work (PoW)*, and can reach an agreement of a sequence of blocks by following the *longest chain rule* (*LCR*). Although remarkably simple, NC enjoys very brilliant properties: it doesn't require participants' identities as a setup — a fully permissionless setting and can work as long as more than half of the computation power controlled by honest participates. Due to this, more than six hundred digital currencies leverage NC or its components, *e.g.* PoW, to maintain consensus [2], [3].

Unfortunately, NC and its variants suffer from low transaction throughput (*e.g.*, 7 TPS[1] in Bitcoin and 20 TPS in Ethereum), resulting from the inherently speed-security tradeoff (*i.e.*, block is generated sequentially and the generation rate needs to be relatively small for security constraints [3]–[7]). Despite the improvements made by [8]–[11], these NC-based protocols are still designed based on this paradigm and can't break the security limits to achieve the optimal throughput, *i.e.*, the maximum supported by the underlying communication network.

Parallel chain represents an exciting breakthrough, which first steps out the design paradigm and pushes blockchains' throughput to the optimal. Surprisingly, parallel chain achieves this with a very simple idea — the whole system's throughput can be promoted by increasing the number of chains. In doing so, there are three main challenges. First, each chain's security won't be influenced as the number of chains increases. This has been well-solved by the m-for-1 PoW [4] (introduced in Sec. II-B).

Second, blockchain is essentially a distributed, append-only ledger that outputs a globally ordered transaction sequence. For example, in NC the main chain, *i.e.*, the longest path in a blocktree structure [1] intrinsically provides a global order of all included blocks, resulting in a sequence of transactions. However, without considering cross-chain hash references (See Sec. II-B) in parallel chain, participants observe multiple blocktrees, forming a blockforest. And there doesn't exist any explicit ordering relationships between mined blocks of different blocktrees because in m-for-1 PoW blocks randomly extends one of the parallel chains. Thus, how to globally order blocks into a sequence is the key to design a secure and high-performance parallel chain system.

To meet the challenge, the recent works [12]–[14] have produced a few elegant designs, among which [12], [13] use one special chain, i.e., "synchronization chain" to either be referenced as a global synchronized clock or reference other chains' blocks in sequence, providing the ordering relationship. However, once the single special chain grows slowly, the whole system's block ordering will be delayed, and transaction latency will increase. In [14], Yu *et al*. adopts a simple predefined sequence to order blocks and propose weighted blocks to balance chains' length. To do this, a new attachment block containing fields $(rank, next\_rank)$ is adopted. However, us-

[1]TPS is short for transaction per second.

ing the pre-defined sequence may ruin the causality of blocks. In other words, a block may be inserted after a future block mined on it into the global block sequences, which provides the possibility for new attack vectors (e.g., a later mined block including the same transactions for winning transaction fees). What's more, the additional attachment blocks increase the protocol's overhead and make the analysis complicated. Their limitations make us believe the ordering issue of the parallel chain hasn't been fully explored.

Third, in parallel chain the blocks concurrently mined by multiple participants in different chains may contain the same transactions more than once, especially those transactions with high fees [1]. It's easy to see the resulted transaction redundancy will decrease the whole systems' effective throughput. More importantly, the Byzantine clients can issue multiple conflicting transactions, which contain the same UTXO as input and have different output, into the network. This is also known as double spending attack in NC. Due to the concurrency of blocks, conflicting transactions will be eventually included in the global ordering block sequence. To solve it, the first transaction is confirmed, whereas the rest of conflicting transactions are invalidated. In other words, clients need to have a full record of transactions to tell confirmed transactions. Note that as blocks are created sequentially in NC, only one of the conflicting transactions can be eventually included into the main chain. The differences lead to that light client, i.e., client just store blocks headers, cannot use SPV to verify transactions anymore. Although the prior works, e.g., transaction sharding in [12] and colored transaction scheduling in [13] can solve redundant transactions, there is no solution for conflicting transactions or feasible design for light clients and SPV. All of those motivate our work.

A. Objective and Contributions

In this technical report , we design a permissionless parallel chain protocol, namely Eunomia, which addresses the total block ordering and conflicting transaction issues. We try to make Eunomia's design modular and leverage the simple and thoroughly analyzed NC [3]–[7] to build each chain instance. It's easy to replace some components, *e.g.* using PoS or Ghost [11], to design a new parallel chain protocol. Eunomia is proved to tolerate up to $1/2$ adversarial computation power and meanwhile, to provide the optimal throughput. Our contributions in this technical report are twofold.

(1) To the best of knowledge, we are the first to use "logical clock" in distributed system [15]–[17] to solve the total block ordering issues in parallel chain. Our adoption is based on the following two observations: i) each chain can be an analogy to one process in a machine, and the included blocks can be transformed into a metaphor of events [15]; ii) blocks' contained hash references, *i.e.*, the logical link constructing blocks into chains, can provide evidence for blocks' causality and synchronize chains' logical clocks. Based on the observations, we propose the "virtual logical clock", which can be used to timestamp blocks and computed according to the directed graph of connected blocks. In our approach, all chains are equal and chains' synchronization doesn't rely on any special chain in a distributed way. In addition, our ordering protocol is shown to be as simple as [14], but order blocks according to their causality without using any additional block. What's more, our protocol combines mining incentive with blocks' ordering, and so any denial of ordering attack will bring economic loss to the attacker.

(2) In a UTXO-based blockchain system, the objects involved in the record can be divided into three levels: UTXO, transaction, and client (listed from bottom to up). A client can issue multiple transactions, and a transaction can contain multiple unspent UTXO as input and create new UTXOs. In this technical report , we propose a new fine-grained UTXO sharding, which is different from the coarse sharding based on transactions and clients. The UTXO sharding allows us to trace and to control clients' UXTO in a more detailed way, and so do well solve the conflicting transaction issue in parallel chain. The new sharding design is shown to SPV-friendly, which can make the light client as secure as it in NC, and meanwhile do not sacrifice too many performances (e.g., transaction latency and protocol overhead).

Report Structure. Sec. II introduces the background, and Sec. III introduces system model and design goals. Sec. IV presents the UTXO sharding design, and Sec. V gives the global ordering protocol. The security analysis is given in Sec. VI. Sec. VIII provides the related work. Finally, Sec. IX concludes the report.

II. Background

In this section, we first provide the necessary background of Bitcoin, then present the extended m-for-1 PoW and finally introduce logical clock.

A. A Primer on Bitcoin

In Bitcoin, there are two types of nodes: clients and miners. Clients can create and send transactions to each other, and miners can collect the received transactions, verify them, pack them in blocks. Each block in Bitcoin contains two components, a block header and a set of transactions. The block header includes a hash value of the previous block, a timestamp, a Merkle tree root of transactions and a nonce (whose role will be explained shortly). Blocks are linked together by the hash references and form a chain structure. The chain structure decides a unique sequence of blocks and further provides a sequence of transactions.

Nakamoto Consensus. Bitcoin relies on NC to reach an agreement of blocks. In NC, all miners obey PoW to generate blocks, in which miners need to find a value of the nonce such that the hash value of the new block has required leading zeros [1]. This puzzle-solving process is often referred to as *mining*. Intuitively, the leading zeros determine the chance of finding a new block in each try. By adjusting the number of leading zeros, the blockchain system can control the block generation

rate and then maintain a stable chain growth, e.g., one block per 10 minutes in Bitcoin

Once a new block is produced, it will be broadcast to the entire network. In the ideal case, a block will arrive all the clients before the next block is produced. If this happens to every block, then each client in the system will have the same chain of blocks. In reality, due to the network delay, a miner may produce a new block *before* he or she receives the previous block, a fork will occur where two "child" blocks share a common "parent" block. In general, each client in the system observes a tree of blocks due to the forking. As a result, each client has to choose the longest path from the tree as the main chain, known as the longest chain rule. The common prefix of all the main chains is called the *system main chain*—a key concept that will be used in our analysis.

UTXO Model. The Unspent Transaction Output (UTXO) model is widely used in Bitcoin and other blockchains systems [18], [19]. In this model, a transaction consumes unspent outputs as inputs from previous valid transactions and creates new unspent outputs that can be used in a future transaction. Specifically, a transaction allows senders to use multiple UTXOs as inputs, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender [1]. It's easy to see that a client owns multiple UTXOs at the same time, and his or her account can be worked out by summing up all the belonging UTXOs, which is different from account/balance model [20].

In this technical report, we leverage the LCR and m-for-1 PoW (i.e., an extension of PoW introduced later) to build each chain instance in Eunomia. This is because NC is very simple and has many promising properties (See Sec. I). More importantly, NC has been thoroughly analyzed in various prior works [3]–[7] and proved to provide the end-to-end security. For the transaction model, our system uses the UTXO model for its simplicity and parallelizability. A client can simultaneously issue multiple transactions with different unspent UTXOs.

B. The m-for-1 PoW

By extending PoW, a miner can simultaneously generate blocks on m chains, and each time a block extends one chain. This extended PoW is known as m-for-1 PoW [4], [12]. See Fig. 1 for an illustration. In m-for-1 PoW, each block contains m hash references, each of which links to the last block in m main chains. Each main chain is independently chosen by certain rules (*e.g.*, LCR in NC and the heaviest subtree rule in the GHOST protocol [11]). When a valid nonce is found, the last $\log_2 m$ bits of the block's hash value can index to one of the m chains that the block belongs to [14]. Here, the hash reference of the block in the belonging chain is called chain reference, and the rest $m-1$ are called cross-chain references. As the hash function can be assumed as a random oracle [4], [6], each chain owns the same chance of receiving a valid block and then maintains the same block generation rate on average. Note that there exist many methods of generating a uniform distribution to randomly assign block to m chains through the random oracle [12], [13]. The resulted same block generation rate ensures each chain owns the same security guarantee. Finally, by adjusting NC's mining difficulty to m times harder, each chain instance in parallel chain can be proved to have the same security guarantee as to the single chain in NC.

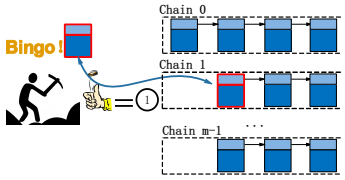

Fig. 1. The m-for-1 PoW for parallel chain

C. Logical Clock

In distributed system, multiple processes running on different entities need to achieve an agreement with the ordering of events, e.g., the sequences of two messages from two different processes, or to avoid simultaneously access the shared resources, e.g., printer [21]. To realize this, in 1978 Leslie Lamport first pointed out that distributed system can use logical clock instead of the physical clock to capture the causal relationship between events [15]. Specifically, the logical clock is an incrementing software counter maintained in each process and synchronized by message exchanges. In [15], he also proposed one realization, called Lamport timestamps, which can provide a partial ordering of events. Furthermore, to realize synchronization of timestamps, Lamport defined a relation called "happens-before": $a \rightarrow b$ means that event a happens before b. Using Lamport's timestamps together with a breaking tie policy, all processes can achieve a total ordering of events. Based on the seminal work, the vector clock is proposed to capture causality between events by Fidge [16] and Mattern [17] in 1988.

Our system uses logical clock instead of vector clock due to its simplicity since the logical clock is easier to be updated and compared. What's more, the vector clock cannot exploit its advantage, i.e., providing correct causality between blocks, in parallel chain because of the forked branches.

III. The System Design and Goals

System Model. Our system model and assumptions follow the well-built formalization of blockchain protocols in [4], [5], [7], [22]. A blockchain protocol Π refers to an algorithm for a set of Turing Machines, also called participants, to interact with each other. The execution of Π is directed by an environment $Z(1^\kappa)$ (where κ is a security parameter), which actives a set of n participants as either being honest for following the protocol, or being corrupt for behaving arbitrarily. Particularly, corrupting participants are assumed to have at most ρ fraction of n participants and are controlled by an adversary $\mathcal{A}$.

The execution of protocol Π proceeds in round; at each round each participant receives messages from Z (*e.g.*, outstanding transactions and other participants' blocks) and makes a query to the random oracle H (*i.e.*, the hash computation for PoW). For each query, participants have a

probability mp to create a new block, where m is the number of parallel chain instances in Eunomia. Here mp is referred to as the *mining hardness parameter* (controlled by the leading zeros in PoW). Particularly, the adversary $\mathcal{A}$ can have up to ρn sequential quires in each round. The NC and Eunomia protocol are referred to as Π_{nak} and Π_{euo}, respectively. The execution of protocol can be modeled as a random variable $\text{EXEC}^{\Pi}(A, Z, \kappa)$, denoting the joint view of all participates (*i.e.*, all their inputs, random coins, and messages received, including those from the random oracle).

Network Model. we assume that honest participants can broadcast messages to each other. $\mathcal{A}$ is responsible for delivering messages (*e.g.*, transactions and blocks) sent by honest participants to all other participants. $\mathcal{A}$ can not modify the content of messages broadcast by honest participants, but it may delay or reorder the delivery of a message as long as it eventually delivers all messages sent by honest participants within some bound Δ. Note that in reality participants are communicating messages through a gossip network, and thus honest participants are assumed to sufficiently connect with each other to guarantee the Δ bounded delay. What's more, to capture the impact of network capacity, we assume the network can transmit at most μ transactions per round on average. Each block size is D transactions.

System Goal. As a blockchain consensus protocol, Eunomia can provide the *safety* and *liveness* guarantees with less than $1/2$ adversaries, and meanwhile have the *optimal* throughput performance. Particularly, in context of blockchain, *safety* corresponds to *consistency* [4], [6], [14]. Thus, Eunomia aims to achieve the followings:

(1) **Consistency.** For any pair of honest participants P_1, P_2 outputting globally confirmed block sequence L_1, L_2 at time $t_1 \leq t_2$, the protocol holds that L_1 is a prefix of L_2.
(2) **Liveness.** All transactions issued by honest clients will eventually get incorporated into an ordered confirmed block in any honest participant's blockchain.
(3) **Near-optimal Network Utilization.** Eunomia aims to achieve a throughput, in terms of transactions processed per second, that approaches the network capacity.

IV. UTXO SHARDING AND SPV

Eunomia composes m parallel chains. Each chain $\mathcal{C}$ is identified with a chain index i ($i \in \{0, 1, ..., m-1\}$) and initialized with a genesis block $\mathcal{G}_i$. The miners adopt LCR and m-for-1 PoW to currently mine on m chains. Each time a mined block $\mathcal{B}$ extends one chain. The index of belonging chain is denoted by $ch(\mathcal{B})$, and the referred block in chain $C_{ch(\mathcal{B})}$ is called parent block. Each block consists of two parts: a block header and a transaction metadata. As shown in Fig. 2, the block header includes a timestamp, a nonce, a hash value of synchronized block and a Merkle tree root of transaction metadata. The reference of synchronized block and transaction metadata do not exist in Bitcoin and will be introduced shortly.

A. UTXO Sharding

In Eunomia, each UTXO contains a $\lceil \log_2(m) \rceil$ bits filed called sharding index. The UTXO's sharding index i ($i \in [0, 1, ..., m-1]$) represents that the UTXO is only allowed to be spent in chain C_i. In other words, a transaction needs to contain the UTXOs with the same sharding index as input, whereas can generate output UTXOs with any sharding index ranging from 0 to $m-1$. Thus, according to input UTXOs' sharding index i, outstanding transactions can be put into corresponding transaction sets S_i ($i \in [0, 1, .., m-1]$). As a result, the transaction tx in $\mathcal{S}_i$ will be mined in chain C_i. It's easy to see that the same transactions or the conflicting transactions cannot be included into several blocks located in different chains, leading to no transaction redundancy or transaction confliction.

B. Transactions Metadata

In Eunomia, participants don't know which chain the future mined block will extend because of m-for-1 PoW. Thus, they need to prepare outstanding transactions and the last blocks' hash value $hash_i$ for each chain $\mathcal{C}_i$. When the number of chain instances m grows larger (e.g., $m = 1000$), the m hash references and the m Merkle roots of transactions directly included into the block header will become an overwhelming overhead.

To solve this, Eunomia leverages the widely-used Merkle Tree [1], [23], [24] to compress these protocol overhead. Specifically, participants first construct transactions for the same chain as a Merkle tree and compute the Merkle tree root γ_i, which is similar to Bitcoin [1]. Then, they use the the pair $(hash_0, \gamma_0)$ through $(hash_{m-1}, \gamma_{m-1})$ as the Merkle tree leaves and compute the related root γ. Finally, the Merkle tree root γ is contained into the block header as input to the PoW puzzle. See Fig. 2 for an illustration. After a block $\mathcal{B}$ is mined, the last $\log_2(m)$ bits of the block's hash value will determine its belonging chain index $ch(\mathcal{B})$. When broadcasting the block, the chosen transaction list, hash references for chain $\mathcal{C}_{ch(\mathcal{B})}$ and the Merkle tree proof of $(hash_{ch(\mathcal{B})}, \gamma_{ch(\mathcal{B})})$ will be attached. When receiving the block, other nodes in the network can verify the included transactions and hash reference for chain $C_{ch(\mathcal{B})}$.

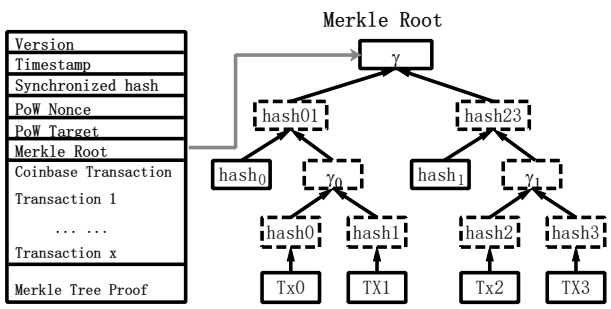

Fig. 2. Data structure of the block and the Merkle tree of transaction metadata.

C. Cross-Chain UTXO

Recall Sec. IV-A, clients are allowed to issue transactions containing the output UTXOs, which have different sharding indexes with the input UTXOs. In other words, these output UXTOs, namely cross-chain UTXOs will be spent in a future transaction located in another chain. By utilizing the cross-chain UTXOs, one one hand clients can collect their owned unspent UTXOs in one chain and combine them for future transactions with a large value input, avoiding issuing multiple splitting transactions in different chains; on the other, clients can freely choose the chain with less outstanding transactions, which can balance chains' transaction workload.

Stale Blocks. For each chain instance in Eunomia, miners have the chance of mining blocks on multiple forked branches, which share the common ancestor blocks. Thus, miners in the system observe m trees of blocks due to the forking. As a result, miners choose a main chain from each tree to mine on according to LCR, and eventually reach an agreement of m system main chains. The blocks included into the system main chain are called regular blocks and the rest are called stale blocks. In Eunomia, only the stale blocks, which provide synchronization references for regular blocks will be kept, and the rest will be discarded (See details in Sec. VII-A.).

However, the stale block may affect the validity of cross-chain UTXOs and the subsequent transactions containing these UTXOs. For example in Figure. 3, if a block A becomes stale and the containing transaction $tx1$ is invalidated, and then a subsequent transaction $tx2$ in block B of another chain, which contains the cross-chain UTXO generated in transaction $tx1$ will be affected. In that case, block B will not be invalidated and just the transaction $tx2$ will be invalidated. Furthermore, all subsequent transactions $tx3$ and $tx4$ that relay on the cross-UTXOs in transaction $tx1$ will become invalidated.

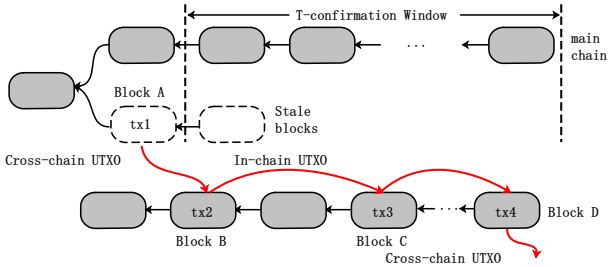

Fig. 3. The infectious window for Cross-chain UTXO due to orphan blocks.

T-confirmation Window. In Eunomia, a block in any chain instance is confirmed with high probability when there are T successive blocks appended. It's also known as T-confirmation (e.g., $T = 6$ in Bitcoin). See Sec. VI for the proved security properties. In other words, after waiting for a T-block growth of the chain, miners can tell whether a block is confirmed or stale with high probability, which we refer it to as T-confirmation window. Thus, to reduce the domino effects of invalidated cross-chain UTXOs, one simple solution is that miners do not include any transactions, which use the cross-chain UTXOs as input, into their blocks until the originate block is confirmed. However, this method will double the latency of the transaction containing input cross-chain UTXOs because of the additional T-confirmation for originate transactions.

To solve this, we allow the subsequent transactions interleaving with the cross-chain UTXOs to be included in the chain first, and the corresponding confirmation is settled later. We call such a confirmation, **Post-Confirmation**. It's easy to see post-confirmation can reduce the transactions' latency. The another reason for adopting post-confirmation is based on the observation that the domino effect of invalidated cross-chain UTXO cannot reach farther with the T-confirmation properties. For example, in Figure. 3, in the T-confirmation window, as miners do not know the transaction $tx1$ and its output cross-chain UTXOs will be invalidated in the future, they can accept the block, which contains the transaction interleaving with the originate cross-chain UTXOs. But when the originate chain grows beyond the T-confirmation window, miners clearly know the cross-chain UTXOs and any transaction interleaving with the cross-chain UTXOs are invalid, and then miners will refuse to accept any blocks containing these invalid transactions. In other words, an invalidated cross-chain UTXOs will only affect the involving transactions in the T-confirmation window.

D. Simple Payment Verification (SPV)

In Eunomia, there are two kinds of clients: full client with a full record of transactions and light client with only block headers, hash references of parent block, and Merkle tree proof for transaction metadata. The hash references of parent block and Merkle tree proof for transaction metadata provide evidence for blocks to be chained. It's easy for a full client to check the validity of transactions by back tracing all input UTXOs in the globally confirmed transaction sequences (which will be introduced shortly). By contrast, because of the invalidated cross-chain UTXOs light clients need to leverage a T-traced verification to realize SPV.

In NC, a light client can use SPV to verify three things: a received transaction is including into the corresponding block; all blocks in the chain have valid proof-of-Work; the block is inserted more than k blocks deeper in the chain, counting from the last block to the corresponding block. The value of k can be set form one to six depending on the transaction value. However, in Eunomia, due to the potentially existed invalidated cross-chain UTXOs, light clients have one more procedure and need to check the validity of the input UTXOs, i.e., checking whether these input UTXOs interleave with some invalidated cross-chain UTXOs. The key idea is utilizing the T-confirmation window. Light client just need to back trace the input UTXOs in T blocks deeper, and ensure miners have confirmed their validity over the T-confirmation.

For a concrete example in Fig. 4. Bob is a light client, who receive a transaction 3 from Alice. He first needs to carry the procedures of SPV in NC to make sure the received transaction is correct except the input UTXOs. Then Bob will verify the inout UTXO of transaction 3, which is the corresponding output UTXO of transaction 2 in Block B. However, block B

is unconfirmed, then Bob will continue verify its input UTXO and eventually find Block A is confirmed.

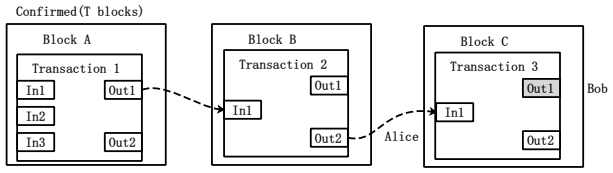

Fig. 4. A Simple Case for chained confirmation for SPV in Eunomia.

V. Global Confirmed Block Ordering

A. Directed graph.

In Eunomia, each node maintains a local view of the directed graph $\mathcal{D}$, depending on its receiving blocks. Each vertex is a block, and each directed edge presents a hash reference of previous block. Upon receiving a new block $\mathcal{B}$, the node will update its local view as follows: 1) Only the block with a valid proof of work will be processed and stored; 2) If the node has the referred parent block and synchronized block (introduced later) in local view, it will compute and verify the block's clock, and then append the block to chain $\mathcal{C}_{ch(\mathcal{B})}$. Otherwise, the block will be cached until parent block or synchronized block are received.

B. Virtual logical clock.

Each chain has a "virtual" clock, which is a counter and increments for each mined block in the chain. Here the virtual logical clock is called because chain's logical clock is not contained in the block field and is computed based on nodes' local view of directed graph $\mathcal{D}$. To synchronize clocks between chains, participants include a hash reference of synchronized block, *i.e.*, the block with the largest virtual logical clock in m main chains, into the pre-mined block's header. As blocks' clock is monotonically increasing in the same chain, synchronized block is one of the m last blocks. It's easy to see that chains' clock synchronizing process is maintained by all participants, *i.e.*, in a distributed way.

When a new block is going to be appended to the nodes' local view, nodes will invoke Algorithm 1 to compute the block's virtual clocks. The algorithm initiate Genesis block $\mathcal{G}_i$'s logical clock as zero. Except Genesis blocks, the algorithm first obtains block's parent block's clock and synchronized block's clock (See line $6-10$). In reality, nodes can cache the latest blocks' clocks in local memory and avoid calling the function repeatedly. The line $10-14$ ensures participants to include the block with the highest clock as synchronized block. Otherwise, their blocks have chance of being timestamped with invalid clock, and then are refused by other nodes, which provides the incentive compatibility for obeying the protocol (addressed later). The virtual clock algorithm is designed as simply as possible, aiming to provide easily proved end-to-end security.

Algorithm 1 logicalClockCompute or LCC in short

1: *Input:* Directed graph $\mathcal{D}$, block $\mathcal{B}$
2: *Output:* Virtual logical clock v of $\mathcal{B} \in \mathbb{R}$
3: **if** $\mathcal{B} \in (\mathcal{G}_i)_{i=0}^{n}$ **then**
4: $v \leftarrow 0$
5: **else** ▷ Synchronize chains' clock
6: $\mathcal{B}_j \leftarrow$ AccessSynchronizedBlocks($\mathcal{D}$, $\mathcal{B}$)
7: v_j = logicalClockCompute($\mathcal{D}$, $\mathcal{B}_j$)
8: $\mathcal{B}_i \leftarrow$ AccessParentsBlocks($\mathcal{D}$, $\mathcal{B}$)
9: v_i = logicalClockCompute($\mathcal{D}$, $\mathcal{B}_i$)
10: **if** $v_i > v_j$ **then** ▷ Incentive compatibility for ordering
11: $v \leftarrow -1$
12: **else**
13: $v \leftarrow v_j + 1$
14: **end if**
15: **end if**
16: **return** v

C. Globally confirmed block sequence.

Each nodes can output a global order on blocks by further using a global logical timestamp (v, i), where v is blocks' virtual logical clock and i represents the blocks' belonging chain index. Specifically, blocks across m main chains can be globally ordered by their clock v with a tie breaking of i.

However, to ensure the consistency property, nodes need further output a global order of confirmed blocks. That implicitly imply that the blocks of each chain participating in global order need to be confirmed first, which we refer to as per-chain confirmed blocks. Specifically, per-chain confirmed blocks are the blocks in each nodes' local chains except the last T blocks according to Theorem 1. For example, a block in Bitcoin usually has to wait for 6 new blocks added to be confirmed. Thus, nodes do not consider the last T unconfirmed blocks of each chain. Let $\mathcal{B}_i$ denote the last per-chain confirmed block in chain C_i with logical clock v_i, and let $synchronized\_bar \leftarrow \min_{i=0}^{m}\{v_i\}$. Then nodes output all the per-chain confirmed blocks whose logical clock is less than $synchronized\_bar$ in the global order, *i.e.*, the globally confirmed block sequence.

For a concrete example, in Fig. 5 we use a round-by-round model to illustrate the block generation and ordering process. In this example, a block reaches all nodes by the end of round and becomes per-chain confirmed after $T = 2$ confirmations. It's easy to see that block $A4$, $B7$ and $C3$ are the last per-chain confirmed blocks, and the $synchronized\_bar$ equals 4. Thus according to the ordering rule, each honest node outputs the total block ordering as: $A1, C1, A3, B1, B3, A4, B5$. Although we present the example using a synchronous model, Eunomia's security will be proved in a asynchronous model introduced later.

D. Incentive Design

Eunomia adopts a similar incentive mechanism as Bitcoin [1], in which participants can receive a block reward, *i.e.*, some

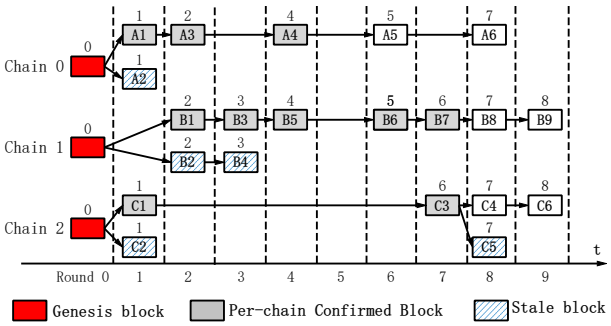

Fig. 5. An illustration of globally confirmed blocks

amount of self-issued tokens, for every mined block included into the globally confirmed blocks. In addition, participants can also get the transaction fees for all the confirmed transactions that are contained in the block. As the incentives in Bitcoin has been well studied [25]–[27], the following focuses on illustrating the incentives in clock synchronization.

In Eunomia, participants concurrently mine on m last blocks of m main chains, and choose one block with the largest virtual clock as the synchronized block. Once a block is mined, it will synchronize its belonging chain's clock with the largest clock, resulting in an increase of $synchronized\_bar$ and global confirmation of blocks. However, some participants may deviate from the protocol and launch ordering attack, *i.e.*, including arbitrary block as the synchronized block. The following analysis show the ordering attack will not have impact with the security of Eunomia, and meanwhile the attacked will suffer from economic loss.

First, if the synchronized block either doesn't exist or doesn't pass the validity check, the referring block will not be processed by Algorithm 1. Second, if the existed synchronized block is valid, but is not the block with the largest clock in the participants' view, there are two possible subcases: 1) Participants mine on m last blocks, but use a block with a smaller logical clock as synchronized block; 2) Participants do not mine on the last blocks in some of the m main chains, but mine on ancestor blocks with a smaller clock. Participants in the first subcase have the chance of making the block invalid for not passing the clock validity check (See line 10-14 in Algorithm 1.), and in the second have the chance that the mined blocks does not belong to the longest chain. It's easy to see both of these cases will make deviating participants suffer from economic loss, which provides incentive compatibility against ordering attack. By contrast, OHIE in [14] lacks effective methods to deal with the ordering attack. What's more, once an honest block included in the chain will synchronize the chain with the latest clock immediately.

VI. Security Analysis

The security analysis of Eunomia is based on several prior works of NC in [4]–[6]. Specifically, the proof composes two steps. First, each chain instance in Eunomia is proved to has the same properties as the single chain in NC. Second, extending each chain's properties further proves *consistency* and *liveness* of Eunomia.

A. Starting Point from NC.

We first give a well-proved corollary from [5], which is the base for our later analysis:

Theorem 1. *(Corollary 3 in [5].) Consider any given constant $\rho < \frac{1}{2}$. Then there exists some positive constant c such that for all $p < \frac{1}{c\Delta n}$, the Π_{nak} satisfies all the following properties in (ρ, Δ, n, p)-environment with probability that drops exponentially in T:*

- *(Consistency) Let S_1 (S_2) be the sequence of blocks on the chain on any Participant P_1 (P_2) at any time t_1 (t_2), excluding the last T blocks on the chain. Then either S_1 is a prefix of S_2 or S_2 is a prefix of S_1.*
- *(Chain-growth) The length of the main chain of any honest participants increases by at least T blocks every $\frac{2T}{pn}$ round.*
- *(Chain-quality) For any T consecutive blocks in any main chain of any honest participants, $(1-\frac{\rho}{1-\rho})T$ blocks were contributed by honest participants.*

Lemma 1. *Consider any give i where $0 \leq i \leq m-1$, the chain $\mathcal{C}_i$ of protocol Π_{enu} in (m, ρ, Δ, n, mp)-environment has the same properties with the single chain of Π_{nak} in (ρ, Δ, n, p)-environment.*

Proof. Without loss of generality, we consider chain C_0 of protocol Π_{enu} and shows it runs in exactly the same way except how the blocks are generated for the chain. In Π_{enu}, each participant has a probability mp to create a new block with $\log(\frac{1}{mp})$ leading zeros, and a probability $\frac{1}{m}$ to has a returned value with $\log(m)$ tailing zeroes in each query to the random oracle. As these two events are independent, miners has a probability p to create a new block for chain C_0, which has $\log(\frac{1}{mp})$ leading zeros and $\log(m)$ tailing zeroes in each query to the random oracle. Thus, chain C_0 has the same probability with the single chain of Π_{nak} to have a new block.

In Π_{enu}, the parents' hash value are included into the transaction metadata Merkle tree, and Merkle tree root is inclued in the block header. Same with Π_{nak}, the parent block can never be reverted once the block is generated. That means the system knows a single parent block. Thus, blocks in chain C_0 are linked just as the single chain of Π_{nak}. □

B. Main Theorem Statements

Eunomia is proved to satisfy the following properties:

Theorem 2. *Consider any given constant $\rho < \frac{1}{2}$. Then there exists some positive constant c and positive integer m such that for all $p < \frac{1}{c\Delta n}$, the Π_{enu} satisfies all the following properties in (m, ρ, Δ, n, mp)-environment with probability that drops exponentially in T:*

- *(Consistency) For any pair of honest participants P_1, P_2 outputting a globally confirmed block sequence L_1, L_2 at time $t_1 \leq t_2$, the protocol holds that L_1 is a prefix of L_2.*

- *(L-growth) For any integer $\gamma \geq 1$, the length of the globally confirmed block sequence L of any honest participants increases by at least $m \cdot \gamma \cdot T$ blocks every $(\gamma+2)\frac{2T}{pn}$ round.*
- *(L-quality) For any $m \cdot \gamma \cdot T$ consecutive blocks in the globally confirmed block sequence L of any honest participants, $m \cdot \gamma \cdot (1 - \frac{\rho}{1-\rho})T$ blocks were contributed by honest participants.*

Theorem 1 shows that the globally confirmed block sequence L satisfies the consistency, quality, and growth properties. Specifically, the L-growth and L-quality states the globally confirmed block sequence L will include honest miners' blocks at a certain rate. Furthermore, clients' transactions are guaranteed to be eventually processed. In other words, consistency corresponds to the safety of Eunomia, while the combination of quality and growth captures the liveness of Eunomia.

Lemma 2. *If the three properties in Theorem 2 hold for each of the m chains of protocol Π_{enu} in (m, ρ, Δ, n, mp)-environment, then the protocol Π_{enu} in (m, ρ, Δ, n, mp)-environment satisfies the consistency property in Theorem 1 with probability that drops exponentially in T.*

Proof. Let the view of participant P_1 at time t_1 be $view_1$, and the view of participant P_2 at time t_2 be $view_2$. Let s_1 (s_2) be the $synchronized\_bar$ in $view_1$ ($view_2$, respectively). Let L_1 (L_2) be the globally confirmed block sequence for $view_1$ ($view_2$, respectively). Let $E_1(i)$ $(0 \leq i \leq m-1)$ be the per-chain confirmed block sequence of chain i in $view_1$, and $G_1(i)$ be the prefix of $E_1(i)$ such that $G_1(i)$ contains all blocks in $E_1(i)$ whose logical clock is smaller than s_1. In the same way, we have $E_2(i)$ and $G_2(i)$. Before proving L_1 is a prefix of L_2, we first give the following claim.

For all i where $0 \leq i \leq m-1$, let $G_1(i)$ is a prefix of $G_2(i)$. According to the **consistency** property in Theorem 1, $E_1(i)$ is a prefix of $E_2(i)$. Let block $B_1(i)$ ($B_2(i)$) be the last block in $E_1(i)$ ($E_2(i)$, respectively). In addition, let $v_1(i)$ ($v_2(i)$) denote the virtual logical clock of $B_1(i)$ ($B_1(i)$, respectively). As blocks' clock monotonically increase in the same chain, $v_1(i) \leq v_2(i)$. By ordering rule, we have $s_1 \leftarrow \min_{i=0}^{m-1}\{v_1(i)\}$ and $s_2 \leftarrow \min_{i=0}^{m-1}\{v_2(i)\}$. Hence, we have $s_1 \leq s_2$. It's easy to see that $G_1(i)$ is a prefix of $G_2(i)$ with $E_1(i)$ is a prefix of $E_2(i)$ and $s_1 \leq s_2$. In addition, any block in $G_2(i) \setminus G_1(i)$ has clock with no less than s_1.

The globally confirmed block sequence L_1 in $view_1$ is constructed by that all blocks in $G_1(0)$ through $G_1(m-1)$ are ordered by their logical clock with a tie breaking of chain index. As $G_1(i)$ is a prefix of $G_2(i)$ for all i $(0 \leq i \leq m-1)$, thus all blocks in L_1 belong to L_2, and are also ordered in the same way. For all block $B \in \bigcup_{i=0}^{m-1}\{G_2(i) \setminus G_1(i)\}$, by the previous claim, B's logical clock must be no smaller than s_1. Thus these blocks must be ordered after all the blocks in $\bigcup_{i=0}^{m-1}\{G_1(i)\}$ (whose logical clock must be smaller than s_1). Thus, it's clear that L_1 is the prefix of L_2. □

Lemma 3. *If the three properties in Theorem 2 hold for each of the m chains of protocol Π_{enu} in (m, ρ, Δ, n, mp)-environment, then a per-chain confirmed block in any chain will become globally confirmed within at most $\frac{4T}{pn}$ rounds with probability that drops exponentially in T.*

Proof. Consider any given honest participant P at given time t_0 has the view $view$. Let $E_i(t_0)$ $(0 \leq i \leq m-1)$ be the per-chain confirmed block sequence of chain i in $view$ at time t_0. With loss of generality, we assume t_0 is after the first $\frac{2T}{pn}$ rounds of the execution, which guarantee there is at least one per-chain confirmed block in $E_i(t_0)$. Let block $B_i(t_0)$ denote the last block in $E_i(t_0)$ at time t_0.

Let x be the largest virtual clock of blocks $\{B_i(t_0)\}_{i=0}^{m-1}$. By time $t_1 = t_0 + \frac{2T}{pn}$, each chain of participant P's $view$ must have at least T blocks attached. By the quality property in Theorem 1, in the T blocks, there must have one block of honest participants which synchronizes its belonging chain to the value x. Then from time t_1 to time $t_2 = t_1 + \frac{2T}{pn}$, the T blocks generated from t_0 to t_1 will become partially confirmed. At time t_2, the $synchronized\_bar$ must be no less than x. Then all per-chain confirmed blocks at time t_0 will become globally confirmed. □

Lemma 4. *If the three properties in Theorem 2 hold for each of the m chains of protocol Π_{enu} in (m, ρ, Δ, n, mp)-environment, then the protocol Π_{enu} in (m, ρ, Δ, n, mp)-environment satisfies the L-quality and L-growth properties in Theorem 1 with probability that drops exponentially in T.*

Proof. Consider any given honest participant P at given time t_0 has the view $view$. Let $E_i(t_0)$ be the per-chain confirmed block sequence of chain i in $view$ at time t_0. For ant integer $\gamma \geq 1$, each chain of the protocol Π_{enu} in (m, ρ, Δ, n, mp)-environment has at least $\gamma \cdot T$ blocks becoming newly partially-confirmed from time t_0 to time $t_1 = t_0 + \gamma\frac{2T}{pn}$. By Lemma 4, we know after $\frac{4T}{pn}$ rounds, any of the per-chain confirmed blocks will become global confirmed. The proof is done. □

VII. Discussion

A. Block Propagation and Stale Blocks

Recall Algorithm 1, the synchronized block must be cached in participants' local view before computing the referring block's clock, which leads to two new issues. First, the referred synchronized block may not be included in main chain, which we refer it to as stale block. In Bitcoin, stale blocks are eventually abandoned by nodes. For a new node joining in the network, it's hard for the node to invoke Algorithm 1. To solve this, nodes can store synchronized blocks' header and the Merkle tree proof of its included hash references, which is similar to uncle blocks in Ethereum [20].

Second, if the synchronized block belongs to a different chain with the referring block, that means nodes' verification and acceptance of this new arriving block will be dependent on a existed cross-chain block. In other words, each chain's block generation can't be viewed as independent with each other any more, which deviates form the design goal and makes analysis

complicated. However, it's not a tough issue. First, most of the nodes will receive the synchronized block before the arriving of the next block mined on it. Second, in Eunomia nodes will relay the synchronized block with newly mined blocks if their neighbors do not have the synchronized block. Thus, each chain's block generation process still remain independent.

VIII. The Related Work

In general, there are three main approaches in the literature to improve the scalability of PoW-based permissionless blockchains. The first is based on the extension of NC. The second is to leverage a more generalized graph (*e.g.*, directed acyclic graph (DAG) and parallel chains) instead of the chain structure. The third is to adopts sharding protocol to scale out.

Extended NC. The Bitcoin community in [28] proposed Bitcoin Unlimited (BU), which breaks NC's fixed block size limit and allows miners to decide the limit value collectively. However, Zhang and Preneel in [8] showed that the absence of block validity consensus (BVC) can lead to new attack vectors. In [9], Eyal *et al*. proposed Bitcoin-NG, a protocol which decouples the transaction processing from the blockchain maintenance, and significantly promotes the throughput. In Bitcoin-NG, the key blocks' owners are entitled to create many micro-blocks consisting of transactions in one epoch. However, as only a single leader is responsible for generating all micro-blocks, it's easy for a leader to launch the censorship attack, and meanwhile to suffer from DoS attacks due to the revealed IP address. In Eunomia, there are multiple concurrent leaders in charge of transaction processing and leaders' IP addressee can't be foreseen by the adversary until a block is mined.

To avoid the single leader's dilemma, some hybrid consensus protocols [18], [19], [29]–[31] are proposed, which combines NC with classical byzantine agreement (BA). Specifically, NC is used to establish committees in a permissionless way, and BA is adopted to reach an agreement of transactions. The hybrid consensus protocols can provide better throughput and instant finality. However, these blockchain protocols usually assume the adversaries control less than $1/3$ of the votes or computation power. Additionally, with the increase of the committee members, the $O(n^2)$ communication complexity for reaching an agreement will become intolerable [32], and the committee reconfiguration will consume a lot of time [18], [19], [30].

Generalized graph structure. The second class of protocols is replacing the NC'S underlying chain structure with some generalized graph. The protocols such as Ghost [11], Phantom [33], Spectre [10], and Conflux [34] construct the blocks in a directed acyclic graph (DAG) by allowing a single block to reference multiple previous blocks. Particularly, Ghost adopts the heaviest subtree rule instead of LCR to choose one main chain. Phantom, Spectre and Conflux use the DAG to define the global order of all blocks. Although these protocols can leverage the forked blocks and promote the throughput, the complicated graph make it hard to provide a formal proof of their safety, liveness and throughout.

Chainweb [35], [36] is the among the first to maintain a set of parallel chains, in which each chain cross-references all other chains according to the base graph. However, Chainweb [35], [36] doesn't provide a formal analysis for its claimed safety and better throughput performance. Based on the work, Kiffer *et al*. [7] analyzed a special case of Chainweb's configuration, in which each chain cross-references all other chains, and showed that Chainweb has the same throughput with the single chain of NC under the same security guarantee. Attracted by the parallel chain's appealing characteristics, there are three recent concurrent work [12]–[14] to propose better protocols. In [12], [13], one of m chains is designed to provide "clock" to synchronize all the blocks. However, the single special chain makes the whole system vulnerable to attacks or is the bottleneck for performance. In [14], Yu *et al*. utilized additional attachment blocks, which is inefficient and incompatible. Particularly, the protocols [13], [14] are analyzed in a synchronous model. Sec. I has discussed these three works in detail.

Sharding-based protocol. In [37], Jiaping Wang and Hao Wang propose a protocol called Monoxide, which composes multiple independent single chain consensus systems, called zones. They also proposed eventual atomicity to ensure transaction atomicity across zones and Chu-ko-nu mining to ensure the effective mining power in each zone. Monoxide is shown to provide $1,000x$ throughput and $2,000x$ capacity over Bitcoin and Ethereum. Except this work, there are some committee-based sharding protocols [18], [30], [38]. Each shard is assigned a subset of the nodes, and nodes run classical byzantine agreement (BA) to reach agreement. However, these protocols can only tolerant up to $1/3$ adversaries. What's more, all sharding-based protocols have additional overhead and latency for cross-shard transactions.

IX. Conclusion

In this technical report, we design Eunomia, a permissionless parallel chain protocol that use virtual logical clock to realize global ordering of blocks. In addition, Eunomia adopts a fin grained UTXO sharding, which do well solve the conflicting transactions issue and is shown to be SPV-friendly. Eunomia is proved to provide the end-to-end safety and liveness guarantees with less than $1/2$ adversarial computation power.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Working Paper*, 2008.

[2] mapofcoins, "Map of coins: Btc map," 2018. [Online]. Available: http://mapofcoins.com/bitcoin

[3] R. Zhang and B. Preneel, "Lay down the common metrics: Evaluating proof-of-work consensus protocols' security," in *Proceedings of the 40th IEEE Symposium on Security and Privacy*, ser. S&P. IEEE, 2019.

[4] J. Garay, A. Kiayias, and N. Leonardos, "The Bitcoin Backbone Protocol: Analysis and Applications," in *Advances in Cryptology - EUROCRYPT 2015*, E. Oswald and M. Fischlin, Eds. Springer Berlin Heidelberg, 2015, pp. 281–310.

[5] R. Pass, L. Seeman, and A. Shelat, "Analysis of the Blockchain Protocol in Asynchronous Networks," in *Advances in Cryptology – EUROCRYPT 2017*, J.-S. Coron and J. B. Nielsen, Eds. Springer International Publishing, 2017, pp. 643–673.

[6] J. Garay, A. Kiayias, and N. Leonardos, "The Bitcoin Backbone Protocol with Chains of Variable Difficulty," in *Advances in Cryptology – CRYPTO 2017*, J. Katz and H. Shacham, Eds. Cham: Springer International Publishing, 2017, pp. 291–323.

[7] 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*, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 729–744.

[8] R. Zhang and B. Preneel, "On the necessity of a prescribed block validity consensus: Analyzing bitcoin unlimited mining protocol," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '17. New York, NY, USA: ACM, 2017, pp. 108–119.

[9] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse, "Bitcoining: A scalable blockchain protocol," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. Santa Clara, CA: USENIX Association, 2016, pp. 45–59.

[10] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "Spectre: A fast and scalable cryptocurrency protocol," *IACR Cryptology ePrint Archive*, vol. 2016, p. 1159, 2016.

[11] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *Financial Cryptography and Data Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 507–527.

[12] M. Fitzi, P. Gazi, A. Kiayias, and A. Russell, "Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition," *Cryptology ePrint Archive, Report 2018/1119*, 2018.

[13] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Deconstructing the blockchain to approach physical limits," *arXiv preprint arXiv:1810.08092*, 2018.

[14] H. Yu, I. Nikolic, R. Hou, and P. Saxena, "Ohie: Blockchain scaling made simple," in *Proceedings of the 41th IEEE Symposium on Security and Privacy*, ser. S&P. SAN FRANCISCO, CA, USA: IEEE, 2020.

[15] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[16] C. J. Fidge, "Timestamps in message-passing systems that preserve the partial ordering," *Proc. of the 11th Australian Computer Science Conference (ACSC'88)*, pp. 56–66, Feb. 1987.

[17] F. Mattern *et al.*, "Virtual time and global states of distributed systems," *Proc. Workshop on Parallel and Distributed Algorithms*, pp. 215–226, 1988.

[18] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*, May 2018, pp. 583–598.

[19] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 17–30.

[20] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, 2014. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[21] A. S. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*. Upper Saddle River, NJ: Pearson Prentice Hall, 2007.

[22] R. Pass and E. Shi, "Rethinking large-scale consensus," in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, Aug 2017, pp. 115–129.

[23] R. C. Merkle, "Protocols for public key cryptosystems," in *1980 IEEE Symposium on Security and Privacy*, April 1980, pp. 122–122.

[24] S. Haber and W. S. Stornetta, "Secure names for bit-strings," in *Proceedings of the 4th ACM Conference on Computer and Communications Security*, ser. CCS '97. New York, NY, USA: ACM, 1997, pp. 28–35.

[25] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *Commun. ACM*, vol. 61, no. 7, pp. 95–102, Jun. 2018.

[26] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in Bitcoin," in *International Conference on Financial Cryptography and Data Security*. Berlin, Heidelberg: Springer, 2016, pp. 515–532.

[27] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. Vienna, Austria: ACM, 2016, pp. 3–16.

[28] T. B. U. C. Organization., "Bitcoin unlimited faq," 2019. [Online]. Available: https://www.bitcoinunlimited.info/faq

[29] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 279–296.

[30] 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*, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 931–948.

[31] R. Pass and E. Shi, "Hybrid Consensus: Efficient Consensus in the Permissionless Model," in *31st International Symposium on Distributed Computing (DISC 2017)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 91. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 39:1–39:16.

[32] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 173–186.

[33] Y. Sompolinsky and A. Zohar, "Phantom: A scalable blockdag protocol." *IACR Cryptology ePrint Archive*, vol. 2018, p. 104, 2018.

[34] C. Li, P. Li, D. Zhou, W. Xu, F. Long, and A. Yao, "Scaling nakamoto consensus to thousands of transactions per second," *arXiv preprint arXiv:1805.03870*, 2018.

[35] S. and Popejoy, "Chainweb: A proof-of-work parallel-chain architecture for massive throughput," 2018. [Online]. Available: https://kadena.io/docs/chainweb-v15.pdf

[36] M. Quaintance and W. Martino, "Chainweb protocol security calculations," 2018. [Online]. Available: https://kadena.io/docs/chainweb_calculations_v7.pdf

[37] J. Wang and H. Wang, "Monoxide: Scale out Blockchains with Asynchronous Consensus Zones," in *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*. Boston, MA: {USENIX} Association, 2019, pp. 95–112.

[38] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 31–42.