Fractal Platform

Thang Dinh, Lei Fan, Jonathan Katz, Hong-Sheng Zhou

Abstract

We introduce Fractal Platform, a high-performance, scalable, provably secure permissionless blockchain. The Fractal system incorporates several components. The first is iChing, a novel proof-of-stake protocol in large-scale network, that is provably secure and addresses notorious challenges such as nothing-at-stake and grinding attacks. The second is BackPackers, a new paradigm for achieving scalability by decoupling data distribution and data ordering; via rigorous analysis and real-world benchmarking, nearly optimal throughput and block-propagation time can be achieved.

Fractal's design has been implemented and tested among 10,000+ nodes across the globe and, under modest network assumptions, sustains a throughput of 3,000+ transactions per second.

Contents

1 Introduction

Bitcoin [13] and blockchain technologies have been widely received. In Bitcoin, a group of participants, *miners*, execute a *consensus protocol* to maintain a growing list of records, called *transactions*. More concretely, transactions are grouped into *blocks*, and then linked via cryptographic hash. To prevent Sybil attacks, Bitcoin protocol uses a *proof-of-work* mechanism: whoever can find a valid solution, to "the proof-of-work puzzle" will become the *block producer* and has the right to generate the next block. The newly generated block will be appended to the longest chain and be broadcast to all other miners via a peer-to-peer (P2P) overlay network. Bitcoin provides the first permissionless consensus protocol which can be executed in a large-scale network. Its security has been intensively investigated and even rigorously analyzed [9, 14].

Unfortunately, Bitcoin (and existing proof-of-work based) systems are facing many challenges. In particular, these systems are extremely slow: less than 7 transactions can be processed in Bitcoin, and these transactions will be confirmed in hours. By carefully choosing parameters such as *block size*, i.e., the maximum size for each block, and *block interval*, i.e., the average time between two consecutive blocks, the performance can be improved but very limited. In our research paper [6], we show that simple tweaking of parameters as a performance improvement solution for Bitcoin is difficult. In the same paper, we also show existing scalability solutions to Bitcoin like systems, without optimizing the utilization of the P2P network can *not* improve the performance of these blockchain systems significantly.

Another major complaints about these proof-of-work based blockchain systems is that they are "environment *un*friendly": Generating a single block on the Bitcoin blockchain requires a large number of hashing operations[1] and maintaining the Bitcoin system wastes a huge amount of energy. Blockchain systems without using proof-of-work has been intensively investigated. For example, many proof-of-stake protocols have been proposed. However, designing a proof-of-stake mechanism for a large-scale network of participants is tricky. Good candidates should be able to defend against many known attacks such as "nothing at stake" and "grinding" attacks, and they should be rigorously analyzed with security proofs.[2]

Fractal's Goals. We intend to solve the above major technical challenges in our Fractal project. Fractal is a blockchain platform that supports from basic payments to multiple advanced functionalities, with the following focuses:

Security From a theoretical point of view, the design must be rigorously analyzed, and the security must be based on realistic assumptions. From a practical point of view, the system should defend against all known attacks.

Scalability The system should achieve high performance. The system should support high-throughput transactions, i.e., the network as a whole should be able to confirm up to ten thousands of transactions per second. In addition, the system should also be low latency, i.e., transactions should be processed and confirmed, within ten seconds.

Sustainability The system should be energy efficient and should not rely on huge amount of computing resources.

In the remaining of the writing, we will describe our strategies to achieve the above goals. In Section 2, via a novel permissionless proof-of-stake protocol, we achieve the goal of sustainability.

[1] https://bitcoinwisdom.com/bitcoin/difficulty

[2] Many proof-of-stake systems use BFT variants; they deviate significantly from Bitcoin's consensus design which involves extremely simple communication structure and low communication complexity. these protocols cannot be executed in a large-scale network with 10,000 participant nodes.

In Section 3, via a set of new strategies to optimizing the utilization of the network for a blockchain system, we achieve the goal of scalability. Security is our first priority, and in the designs in both sections security have been achieved with rigorous analysis under very realistic assumptions.

We note that, in this writeup, we intend to provide an accessible description for our designs in the consensus layer (layer 1) and in the network layer (layer 0). Fractal's design has been implemented and tested among 10,000+ nodes across the globe and, under modest network assumptions, sustains a throughput of 3,000+ transactions per second. The implementation and testing will be described in a separate writeup.

2 Layer 1: Secure and Permissionless Proof-of-Stake

In this section, we will first review Nakamoto's consensus design in Bitcoin system. Then we will present our iChing consensus. Similar to Bitcoin, our iChing is a permissionless consensus protocol for large-scale network. Different from Bitcoin, our iChing is based on a *proof-of-stake* mechanism; that means, our protocol does not require huge amount of electricity and dedicated computing hardware. Many design/analysis details can be found in our research paper [8]. We note that, our design is the first (and only known) natural mimic of Bitcoin via proof of stake: the blocks are extended in the "per block" sense; the next block generator is defined only when the current block is published.[3]

2.1 Nakamoto's Design

We first briefly review Nakamoto's design ideas [13]. The blockchain in Bitcoin consists of a chain of ordered blocks $B_1, B_2, B_3, \ldots$, and PoW-players (i.e., miners) in each round (or time slot) attempt to extend the blockchain with a new block by solving proof-of-work puzzles [1, 7]. The *puzzle* for each miner is defined by (1) the "context", i.e., the latest block in the longest blockchain in the miner's view, and (2) the "payload", i.e., the set of valid transactions to be included in the new block; and a valid *puzzle solution* to the problem is defined by a hash inequality. More concretely, assume the longest blockchain for a miner consists of ordered blocks $B_1, B_2, \ldots, B_i$, and B_i is the latest block. The miner now attempts to find a valid puzzle solution *nonce* which can satisfy the following hash inequality: $\mathsf{H}(\mathsf{hash}(B_i), \mathit{payload}, \mathit{nonce}) < \mathtt{T}$, where $\mathsf{H}(\cdot)$ and $\mathsf{hash}(\cdot)$ are two hash functions, $\mathit{payload}$ denotes the set of valid transactions to be included in the new block, and $\mathtt{T}$ denotes the target of proof-of-work puzzle difficulty (which specifies how difficult to identify a puzzle solution by making a hash query attempt). In the case that a new valid solution, *nonce*, is identified, such a solution can be used for defining a new valid block B_{i+1} as follows: $B_{i+1} := \langle h_i, \mathit{payload}, \mathit{nonce}\rangle$, where $h_i := \mathsf{hash}(B_i)$. Then the new block B_{i+1} will be revealed by the miner, and broadcasted to the network and then accepted by the remaining miners in the system. (The above description is oversimplified.)

We may consider a further simplified version of the above blockchain protocol, called *Bitcoin core-chain protocol*. In the core-chain protocol, the payload will be ignored, and now puzzle is based on $\mathsf{H}(\mathsf{hash}(B_i), \mathit{nonce}) < \mathtt{T}$, and the new block B_{i+1} is defined as $B_{i+1} := \langle h_i, \mathit{nonce}\rangle$.

[3] In a concurrent interesting proposal [2, 5], blocks are extended in the "per epoch" manner; there, the next block generator is defined an epoch (i.e., many blocks) ago.

Algorithm 1: Block generation in Bitcoin

Initialize chain with the genesis block;
while *true* **do**
  $\mathcal{C}_{\text{best}} \leftarrow$ the longest chain in the local state of the miner;
  $prev \leftarrow$ hash value of the last block in $\mathcal{C}_{\text{best}}$;
  $nonce \leftarrow$ a random number;
  if $\mathsf{H}(prev, nonce) < \mathtt{T}$ **then**
    Create new block $B = \langle prev, nonce \rangle$;
    Broadcast B;

2.2 iChing Design

2.2.1 The basic version

We intend to mimic Nakamoto's design but via proof-of-stake. We now describe the basic version of our iChing design; this basic version will be further improved in next subsections.

In addition to the hash functions that have been used in Nakamoto's design, in iChing, we need an additional building block, *unique digital signature scheme* [4, 12] $(\mathsf{uKeyGen}, \mathsf{uKeyVer}, \mathsf{uSign}, \mathsf{uVerify})$. All proof-of-stake players must have themselves registered to the system; each player holds his signing-verification key pair (SK, PK).

In the basic version here, we consider the strategy that all players attempt to extend the longest chain with a new block. Similar to that in the PoW-based protocol, a winning player is chosen with some probability but using a different hash inequality. More concretely, assume the longest chain for a player consists of the following ordered blocks, $B_1, B_2, \ldots, B_i$; let r denote the current time step (or round number); If the player who holds keypair (SK, PK), is chosen, then the following hash inequality holds: $\mathsf{H}(\mathsf{hash}(B_i), r, \text{PK}, \varsigma) < \mathtt{T}$, where $\varsigma := \mathsf{uSign}(\text{SK}, \langle h_i, r, \text{PK} \rangle)$, and $h_i := \mathsf{hash}(B_i)$. The new block B_{i+1} is defined as $B_{i+1} := \langle h_i, r, \text{PK}, \varsigma \rangle$. Please see Algorithm 2 for more details.

Algorithm 2: Block generation in iChing, following the basic strategy

Initialize chain with the genesis block;
$r \leftarrow 1$;
while `true` **do**
  $\mathcal{C}_{\text{best}} \leftarrow$ the longest chain in the local state of the miner;
  $prev \leftarrow$ hash value of the last block in $\mathcal{C}_{\text{best}}$;
  $r \leftarrow$ current round;
  $\varsigma \leftarrow \mathsf{uSign}(\text{SK}, \langle prev, r, \text{PK} \rangle)$;
  if $\mathsf{H}(prev, r, \text{PK}, \sigma) < \mathtt{T}$ **then**
    Create new block $B = \langle prev, r, \text{PK}, \varsigma \rangle$;
    Broadcast B;
  Wait(T_{round}); *Here T_{round} denotes a unit of time for a round*
  $r \leftarrow r + 1$

Our design is very similar to Nakamoto's: the context here consists of the latest block in the longest chain, and the payload in the chain is empty; the puzzle solution consists of the current time, a player's verification key and his signature of the context. When the adversary (1) follows the basic strategy, i.e., extending the single longest chain, and (2) has all stakes registered without being aware of the state of protocol execution, then our protocol can be viewed as a proof-of-stake analogy of Nakamoto's, and the security properties i.e., chain growth, chain quality, and common prefix (cf [9, 14]) can be demonstrated.

2.2.2 Addressing "nothing at stake"

For simplicity, in the basic protocol above, we focus on the setting that all players follow the basic strategy to extend the chain. That is, each player will make attempts to extend the *single* best chain (i.e., the longest chain) in his/her local view. We note that, this basic strategy has been widely adopted in the proof-of-work setting; there, extending a chain is expensive in the sense that it requires significant amount of computing power; it will be extremely costly to extend multiple chains simultaneously. However, in the proof-of-stake setting, it is very cheap to extend a chain. The proof-of-stake players may follow a *full-greedy* strategy to extend all chains, expecting to obtain additional advantage for extending the best chain. This introduces difficulty for security analysis. Furthermore, it is extremely challenging to defend against an adversary who may play an *arbitrary* strategy in the protocol execution. For example, instead of playing the full-greedy strategy to extend *all* chains, the adversary may on purpose extend only the weaker/shorter chains; this may create a scenario that, two (or multiple) chains may take turns to be the best chain, and the common prefix property may not be achieved.

Understanding the power of greedy strategies. To address the above difficulty, our key observation is that, *honest players should also play a carefully designed greedy strategy.* We have two major contributions for understanding the greedy strategies. First, we focus on full-greedy strategy, and demonstrate a very interesting upper bound: the full-greedy strategy will allow a proof-of-stake player to improve his/her chance of extending chains with a factor at most e where $e \approx 2.718$. This upper bound allows us to develop secure chain protocols against full-greedy adversaries.

Defending against an arbitrary adversary. Our second major contribution is, we for the first time shed light on addressing arbitrary adversarial attacks. As discussed before, an arbitrary adversary could break the common prefix property if the blockchain protocol is not carefully designed.

To defend against this powerful adversary, a carefully designed greedy strategy is needed. We introduce a simple but novel, "D-distance-greedy" strategy. A D-distance-greedy player will make attempts to extend a *set of chains*; these chains have the following unique property: after removing the last D blocks, those chains are prefix of the best chain. Note that, by following the D-distance-greedy strategy, the honest miners extend the chain set that share the same prefix. The D-distance-greedy strategy can have common prefix property enabled, which can effectively defend against an arbitrary adversary. If the majority of stakes are honest , then the protocol can achieve the security properties.

Distance-greedy strategies. We first introduce *distance-greedy strategies* for honest protocol players. Consider a blockchain protocol execution. In each player's local view, there are multiple chains, which can be viewed as a tree. More concretely, the genesis block is the root of the tree, and each path from the root to a leaf is essentially a chain. The tree will "grow": the length of each existing chain may increase, and new chains may be created, round after round.

Before giving the formal definition for distance-greedy strategies, we define the "distance" between two chains in a tree.

Distance between two chains. Consider two chains $\mathcal{C}$ with length ℓ, and $\mathcal{C}'$ with length ℓ', respectively. Let d be a non-negative integer. We say the distance of chain $\mathcal{C}'$ from chain $\mathcal{C}$ is d, if d is the smallest non-negative integer so that $\mathcal{C}'[1, \ell' - d] \preceq \mathcal{C}$, and we write $\mathsf{distance}(\mathcal{C} \to \mathcal{C}') = d$.

We note that the distance of $\mathcal{C}$ from $\mathcal{C}'$ is different from the distance of $\mathcal{C}'$ from $\mathcal{C}$, and it is very possible that $\mathsf{distance}(\mathcal{C}' \to \mathcal{C}) \neq \mathsf{distance}(\mathcal{C} \to \mathcal{C}')$. For example, in Figure 1, the distance of $\mathcal{C}$ from $\mathcal{C}'$ is 2, i.e. $\mathsf{distance}(\mathcal{C}' \to \mathcal{C}) = 2$, while the distance of $\mathcal{C}'$ from $\mathcal{C}$ is 4, i.e., $\mathsf{distance}(\mathcal{C} \to \mathcal{C}') = 4$. We also note that the distance of $\mathcal{C}$ from itself is always 0, i.e., $\mathsf{distance}(\mathcal{C} \to \mathcal{C}) = 0$.

Distance-greedy strategy. We are ready to introduce the distance-greedy strategies. Intuitively, a player who plays a distance-greedy strategy will make attempts to extend a *set of chains* in which

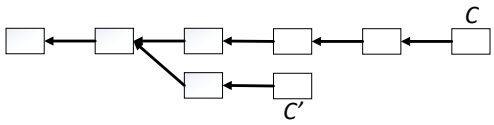

Figure 1 The distance between two chains $\mathcal{C}$ and $\mathcal{C}'$.

all chains have the following properties: (1) the chain should be very "close" to the best chain, i.e., the distance from the chain to the best chain must be small; (2) the chain should not fall behind the best chain too much, i.e., the length of the chain must be big. More concretely, consider a blockchain protocol execution. Let P be a player of the protocol execution, and T be a tree which consists of chains with the same genesis block, in player P's local view. Let $\mathcal{C}_{\text{best}}$ be the longest chain at round r, and parameter $\mathtt{D}$ is a non-negative integer. Define[4] a chain set $\mathbb{C}_{\text{best}}$ as

$$\mathbb{C}_{\text{best}} = \left\{ \mathcal{C} \;\middle|\; \mathsf{distance}(\mathcal{C}_{\text{best}} \to \mathcal{C}) \leq \mathtt{D} \right\}$$

We say the player is $\mathtt{D}$-*distance-greedy* if, for all r, the player makes attempts to extend all chains $\mathcal{C} \in \mathbb{C}_{\text{best}}$.

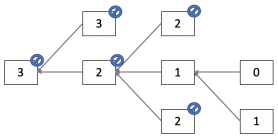

Figure 2 1-distance greedy

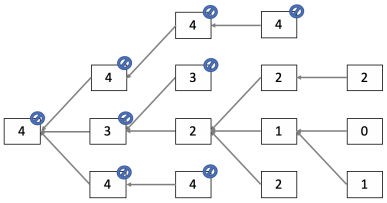

Figure 3 2-distance greedy

In Figures 2 and 3, pictorial illustration for the examples of 1-distance-greedy and 2-distance-greedy strategies can be found respectively. In both figures, each block and all its ancestor blocks together form a chain; the distance of this chain from the best chain will be labeled with this block. All blocks marked with a blue "⊘" symbol on the top right, do not belong to the best chain set, and they not extended by the honest player in the current round.

Once the $\mathtt{D}$-distance-greedy strategy is defined, we can have the following block generation algorithm for iChing protocol.

2.2.3 How to support joining

The protocol above is expected to be executed in a less realistic setting where all players must have their stakes registered without being aware of the state of the protocol execution. Recall that the hash inequality $\mathsf{H}(\mathit{context}, \mathit{solution}) < \mathsf{T}$ is used in the process of extending the chains. In reality, an adversary may have a stake registered *based on the state* of the protocol execution. More concretely, the adversary can play a "rejection re-sampling" strategy to generate keys, and then have his/her stake registered adaptively: the adversary first runs the key generation algorithm to obtain a key-pair $(\mathsf{PK}, \mathsf{SK})$, and then checks if the corresponding (PK, σ) is a valid solution to the hash inequality; if not, the adversary re-samples a new key-pair. This adaptive stake registration

[4] There are multiple variants for defining the best set. See the research paper [8] for more discussions.

Algorithm 3: Block generation in iChing, following the D-greedy strategy

Initialize chain with the genesis block;
$r \leftarrow 1$;
while `true` **do**
  $\mathcal{C}_{\text{best}} \leftarrow$ the longest chain in the local state of the miner;
  $r \leftarrow$ current round;
  $\varsigma \leftarrow \mathsf{uSign}(\text{SK}, \langle prev, r, \text{PK}\rangle)$;
  for *all chain* $\mathcal{C}$ *such that* $\mathsf{distance}(\mathcal{C}_{\text{best}} \rightarrow \mathcal{C}) \leq \mathtt{D}$ **do**
    $prev \leftarrow$ hash value of the last block in $\mathcal{C}$;
    if $\mathsf{H}(prev, r, \text{PK}, \varsigma) < \mathtt{T}$ **then**
      Create new block $B = \langle prev, r, \text{PK}, \varsigma\rangle$;
      Broadcast B;
  Wait (T_{round});
  $r \leftarrow r + 1$

strategy enables the adversary (to be selected) to extend the chains with much higher probability. To address this concern, we introduce new ideas to our protocol design: to extend the chains with new blocks, a player must have his/her stake registered a specified number of rounds earlier.

2.3 Extensions

In this section, we Our design is a natural mimic of Nakamoto's but via proof-of-stake. Therefore, we can easily extend our design above in multiple directions, including enabling adaptive difficulty adjustment and supporting light clients as in the original Bitcoin [13], and incentivizing the system via better strategies (e.g., [15]), managing transactions in blockchain more effectively (e.g., [16]), and more.

2.3.1 How to add transactions back

In the design in Section 2.2, for simplicity, we eliminate the block payload (e.g., the transactions). Now we show how to add transactions back. More concretely, once a new block-core B_{i+1} is generated by a player (in the core-chain protocol), then the player is selected for generating the new block $\tilde{B}_{i+1}$, in the following format: $\tilde{B}_{i+1} = \langle \mathsf{hash}(\tilde{B}_i), B_{i+1}, \hat{X}_{i+1}, \hat{\text{PK}}, \hat{\varsigma}\rangle$, where $\hat{\varsigma} \leftarrow \mathsf{Sign}(\hat{\text{SK}}, \langle \tilde{h}_i, B_{i+1}, \hat{X}_i, \hat{\text{PK}}\rangle)$, $\hat{X}_{i+1}$ is payload and $\tilde{h}_i := \mathsf{hash}(\tilde{B}_i)$, and $B_{i+1} := \langle h_i, r, \text{PK}, \varsigma\rangle$. Here the player holds two pairs of keys, i.e., (SK, PK) of the unique signature scheme $(\mathsf{uKeyGen}, \mathsf{uSign}, \mathsf{uVerify})$, and $(\hat{\text{SK}}, \hat{\text{PK}})$ of a regular[5] digital signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. Now we attach each block to the core-chain via the corresponding block-core; we can reduce the security of the blockchain protocol to the security of the core-chain protocol. Please also see Figure 4 for a pictorial illustration.

2.3.2 How to adjust difficulty adaptively

In Bitcoin, in order to maintain a steady chain growth rate, the system adjusts the PoW hash target difficulty adaptively. The smaller the target, the lower the probability to get a valid PoW block by a hash function query, and vice versa. Our scheme can be extended to support adaptive difficulty

[5] We note, to achieve fully adaptive security, the regular signature scheme should be upgraded into a forward-secure digital signature scheme [3]. We further note that, in the design for blockchain without payload in the previous subsection, we do not use any forward-secure signature scheme.

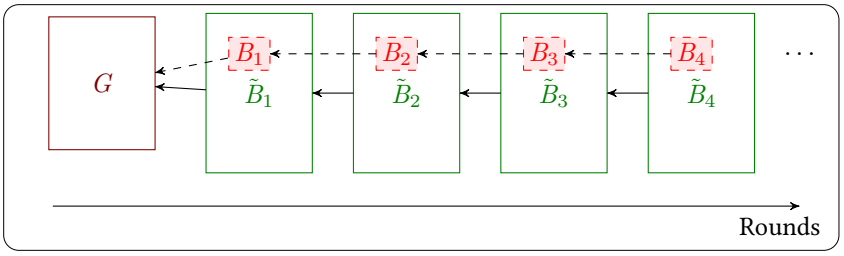

Figure 4 Blockchain structure

Blockchain $\tilde{\mathcal{C}}$ consists of initial setup information (i.e., genesis block) G, and then an ordered sequence of blocks $\tilde{B}_1, \tilde{B}_2, \tilde{B}_3, \ldots$. Here, each block $\tilde{B}_i$ consists of B_i and additional information. A core-chain $\mathcal{C}$ consists of the initial setup information G and the ordered sequence of blocks $B_1, B_2, B_3, \ldots$.

easily. As in Nakamoto's system, the target difficulty is adjusted every m blocks for some integer m. The time span of difficulty adjustment is called an *epoch*; and let $\mathtt{t}$ be the expected time of an epoch. Let t_i be the actual time span of the i-th epoch, and $\mathtt{T}_i$ be the target difficulty in the i-th epoch. We have the target difficulty in the $(i+1)$-th epoch as follows: $\mathtt{T}_{i+1} = \frac{t_i}{\mathtt{t}}\mathtt{T}_i$. From the equation above we can observe that, if $t_i > \mathtt{t}$ then $\mathtt{T}_{i+1} > \mathtt{T}_i$ and vice-versa. In the case that $t_i > \mathtt{t}$, the stakeholders spend longer time to obtain m blocks; it means the system requires more time than expected for the i-th epoch; thus, the target difficulty should be increased so that the stakeholders can find new blocks faster in the next epoch. This *negative feedback* mechanism makes the system stable. To extend a proof-of-stake blockchain, we modify the hash inequality as $\mathsf{H}(\mathsf{hash}(B_i), r, \mathrm{PK}, \varsigma) < \mathtt{T}_i$. A player will test if he is qualified to sign a proof-of-stake block based on the current target difficulty $\mathtt{T}_i$.

2.3.3 Blockchain in the non-flat model

In our main design in Section 2.2, for simplicity we describe our protocol in the "flat" model, where all player s are assumed to hold the same number of stakes (and they are selected as the winning player with the same probability in each round). In reality, player s have different amounts of stake. We next discuss how to extend our design ideas properly into this more realistic "non-flat" model. Consider a player , with verification-signing key pair $(\mathrm{PK}, \mathrm{SK})$, holding v number of stakes. Let $\mathtt{T}_j$ denote the target difficulty in the current epoch, i.e., the j-th epoch. We change the hash inequality as follows:

$$\mathsf{H}(\mathsf{hash}(B_i), r, \mathrm{PK}, \varsigma) < v\mathtt{T}_i$$

It is easy to argue that the winning probability of a player for generating a new block-core is proportional to the amount of stake he controls.

2.3.4 How to achieve post-quantum security

We note that a proof-of-stake player will rely on a blockchain with bounded number of blocks. Let ℓ be the bound, and next we can estimate that $\ell = 2^{22}$ which is not very big. In [10], we consider to use a *relaxed* version of unique signature for constructing proof-of-stake protocols. More concretely, we introduce an $\mathcal{M}$-bounded unique signature scheme in which the security properties including uniqueness and unforgeability hold only for the message space $\mathcal{M}$ (and undefined otherwise). Interestingly, this bounded unique signature scheme can be constructed by utilizing hash functions, and we can achieve post-quantum security.

3 Layer 0: High Performance

The tradeoff between security and performance has been intensively investigated in Bitcoin-like blockchain systems. There exist many scalability proposals. Unfortunately, most of them achieve high performance at the price of security; the remaining only achieve very limited amount performance improvement. The bottleneck of performance in blockchain systems is the extremely inefficient way of utilizing the peer-to-peer network communication.

In this section, we will first present our scalability proposal called BackPackers, in subsection 3.1, to address this bottleneck; then in subsection 3.2, we will present several optimization strategies for BackPackers. We note that, many design/analysis details can be found in our research paper [6].

3.1 BackPackers Design

In our design, a new role of players, called *packers*, are introduced, along with users and miners. While the security of the consensus relies on the miners, these packers can significantly facilitate the data communication and distribution. The very high-level ideas are described in Figure 5.

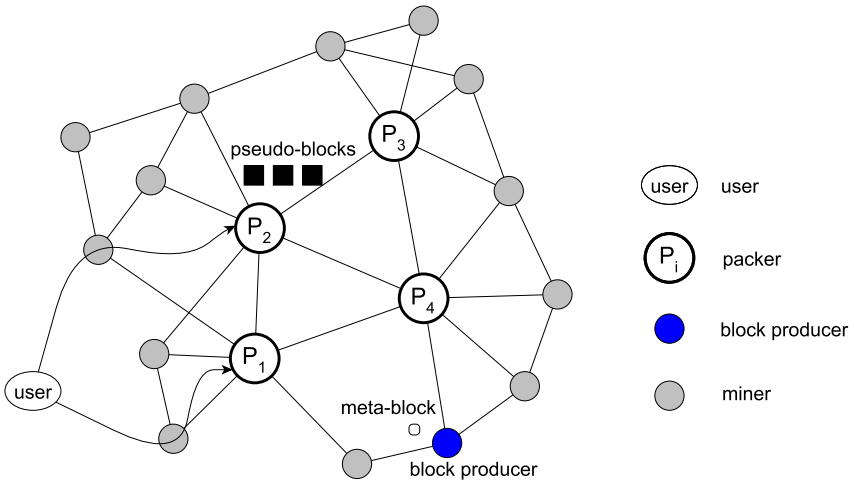

Figure 5 BackPackers paradigm. 1) Users route transactions to all packers; 2) Packers, collect and pack transactions into pseudo-blocks and distribute the pseudo-blocks to miners; 3) Block producer (winning miner) after finding a solution, generates a full-block, and then the block producer broadcasts the corresponding tiny meta-block (and all miners after receiving the meta-block reconstruct the full block).

In our design a set of packers are registered into the system. First, instead of broadcasting transactions into the system as in Bitcoin, users follow a much more efficient way, routing the transactions to all packers. Then the packers choose a set of transactions and pack them into pseudo-blocks; these pseudo-blocks will then be broadcast to the miners.

The data blocks in the blockchain are generated by the miners based on a subset of the transactions in the pseudo-blocks. More concretely, a block producer (winning miner) after finding a solution, generates a full-block, and then the block producer broadcasts the corresponding tiny meta-block (and all miners after receiving the meta-block reconstruct the full block).

3.1.1 Packers

In this part, we describe how packers work in our design. First, each packer must have itself registered, before participating in the process of packing transactions. More concretely, a player can broadcast a registration message in the format: $\langle \mathsf{register}, pk, \varpi \rangle$, where pk is the public key of the packer and ϖ is a proof that the player has the right to be packer. The packer holds the corresponding signing key sk. After the registration, this packer can use sk to generate valid pseudo-blocks under its own pk. We denote (pk_i, sk_i) is the keys pair of packer $\mathtt{P}_i$.

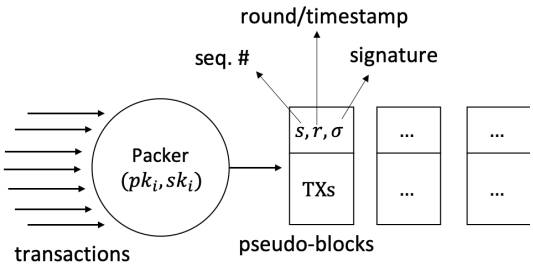

Figure 6 Packers collect transactions from users and broadcast pseudo-blocks to the miners.

Packing transactions Next we describe how a packer packs a set of transactions into a pseudo-block. Please see Figure 6 for a pictorial illustration. Consider a packer $\mathtt{P}_i$ with already registered key-pair (pk_i, sk_i). The packer collects a set of transactions TXs from mempool, and defines a header which consists of the timestamp (i.e., the current round number) and an incremental sequence number. The packer signs the transactions along with the header, and obtains the pseudo-block. Please see Algorithm 4 for details.

Algorithm 4: Pseudo-block generation in BackPackers

$s \leftarrow 1$; ▷ Initialize sequence number.
while `true` **do**
  $r \leftarrow$ current round;
  Select a set of transactions TXs;
  $\sigma \leftarrow \mathsf{Sign}(sk_i, \langle pk_i, s, r, \mathrm{TXs} \rangle)$;
  $\tilde{B} \leftarrow \langle pk_i, s, r, \mathrm{TXs}, \sigma \rangle$;
  Broadcast $\tilde{B}$;
  Wait ($T_{\text{pack-delay}}$);
  $s \leftarrow s + 1$;

3.1.2 Miners

We now describe how miners generate blocks in our BackPackers design; see Figure 7 for a pictorial illustration.

Full-block generation. From the received pseudo-blocks, each miner chooses a subset of pseudo-blocks $\tilde{\mathcal{B}} = \tilde{B}_1 \| \tilde{B}_2 \| \cdots$. Then the miner linearizes all chosen pseudo-blocks and defines the ordered transactions into TXs; let $succinct(\tilde{\mathcal{B}})$ and $succinct(\mathrm{TXs})$ denote the linearization and ordering information, respectively. Now the miner can define the corresponding meta-block by computing the puzzle solution. Once meta-block $meta$ is defined, the full block B is also defined. Please see Algorithm 5 for details.

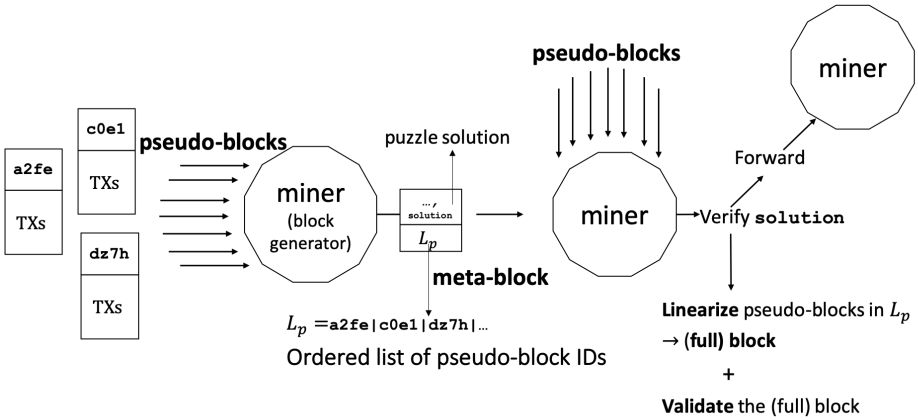

Figure 7 The winning miner, i.e., the block generator creates a full-block and the corresponding meta-block. The winning miner then broadcast the meta-block in the network.

Algorithm 5: Block generation in BackPackers

```
Initialize chain with the genesis block;
while true do
    C_best ← the longest chain in the local state of the miner;
    prev ← hash value of the last block in C_best;
    B̃ ← a set of pseudo-blocks;
    TXs ← the set of transactions that is linearized from B̃;
    Find the solution;
    if H(prev, succinct(B̃), succinct(TXs), solution) < T then
        meta ← ⟨prev, succinct(B̃), succinct(TXs), solution⟩;
        Create new block B = ⟨meta, B̃, TXs⟩;
        Broadcast meta;
```

Full-block reconstruction. In our design, once a meta-block is successfully defined, then the corresponding full-block is defined. As described above, the winning miner generates a full block B and the corresponding meta-block $meta$, and then broadcasts the lightweight meta-block $meta$ to the system. All the remaining miners after receiving the meta-block $meta$, if $meta$ is valid, then the miners will reconstruct the corresponding full-block. Please see Algorithm 6 for details.

From the meta-block $meta$, each miner obtains the ordered list of pseudo-block identifiers; based on the set of pseudo-blocks the miner received before, he can reconstruct the sequence of pseudo-blocks $\tilde{\mathcal{B}}$ that have been used for defining full block B. Recall that the full block B consists of $\langle meta, \mathrm{TXs}, \tilde{\mathcal{B}} \rangle$. To obtain TXs, the miner iterates through all pseudo-blocks $\tilde{B} \in \tilde{\mathcal{B}}$, and be sure in all transactions in these pseudo-blocks are valid and not conflict with existing transactions. When all verifications succeed, the miner reconstructs the corresponding full block B (and update his own local blockchain by including this B).

Algorithm 6: Block reconstruction in BackPackers

```
upon receiving a new meta-block meta do
    Obtain succinct(B̃) from meta;
    Obtain B̃ from succinct(B̃) and the pseudo-blocks;
    TXs ← the set of transactions that is linearized from B̃;
    Reconstruct block B = ⟨meta, B̃, TXs⟩;
```

3.2 Optimization

In the previous subsection we present our basic design of BackPackers. In this subsection, more ideas will be introduced; our goal here is to provide a near optimal solution to high-performant permissionless blockchain in a large-scale network.

3.2.1 Theoretical limits

The players in Bitcoin-like protocols are connected via a peer-to-peer network. We here establish *protocol-agnostic* theoretical limits on the throughput (number of transactions per second) and the block propagation time.

Network limits on the throughput. By abstracting the protocol players as nodes in a peer-to-peer file synchronization system, the result in [11] helps us to identify the two network limits on throughput:

1. *Source bandwidth*: The upload bandwidth of the block producers will be a limit on the throughput.
2. *Average upload bandwidth*: As all nodes need to receive the same blocks, even when all nodes are sending the data to the other, the throughput cannot exceed this network limit.

Network limit on the propagation time. The block propagation time is one of the deciding factor for confirmation time and the security in general. In addition to the time to verify the transactions, the major theoretical limit on the propagation time is the *maximum latency path* among the nodes in the network. That is the time to transmit a zero-size block to each node along the "shortest-paths" in the network.

3.2.2 Near-optimal performance

The above network limits define the optimal performance for blockchain protocols. Next, we describe how we achieve near-optimal throughput and block propagation time, respectively.

Achieving near-optimal throughput. We first analyze the existing scaling proposals for Bitcoin, to find out the main culprits that affect the throughput. Typically, in blockchain protocols, the data (e.g., transactions, blocks) are propagated by a three-step scheme: 1) nodes send invite messages to the neighbors announce the knowledge of new data; 2) nodes send getdata messages to request data after receiving invite messages; 3) nodes send data responses to getdata messages. There are two major performance culprits in the three-step scheme:

- *High transaction relay overhead* that is much higher than the actually transmitted data;
- *Inefficient block propagation scheme* in which some nodes are overloaded with transmission requests while other nodes' bandwidths is unused.

Since the packers gather the transactions from users and only broadcast pseudo-block with bigger size. The relay overhead now become trivial. Additionally, we propose *throughput-optimal propagation* scheme to balance the overload among the nodes in the network. We balance the workload among nodes by using a backpressure method, i.e.,

- nodes send requests to the neighbor with minimum pressure (i.e., the node that has to serve the smallest number of requests);
- nodes serve request from the neighbor with maximum pressure (i.e., the node that sent the highest number of requests).

▶ Remark (Soft Spatial Sharding). Additionally, we introduce a strategy called *soft spatial sharding*, for the packers to avoid including duplicate transactions, to maximize the throughput. In particular, we introduce a ranking function which can determine for each given transaction, the best packer, the second best packer, and the n-th best packer, among a set of packers. Based on the rank, the packers are allocated corresponding time windows; if the transaction is not packed by the packers with high priority, then the next best packer will pack that transaction. By using this strategy, any transaction will be packed by at least one packer and the honest packers will not pack duplicate transactions.

Achieving near-optimal propagation time. Our key to achieving near-optimal block propagation time is that miners are only required to propagate the light-weight meta-blocks. The miners unsolicited send to their neighbor after verifying the hash inequality. By this way, the meta-block is propagated through the shortest path. At the same time, the adversary cannot spam the network by providing invalid meta-blocks: for each valid meta-block, a non-trivial solution[6] must be included. Note that, the size of meta-block is tiny and the verification can be done instantly; the meta-block can be propagated to all players in network in near-optimal time.

Furthermore, we enforce the winning miners to take an intentional packing delay. That is, the winning miners only include the pseudo-blocks that already received by all other miners. Thus, the miners can reconstruct the full-block when they receive the meta-block.

References

1 Adam Back. Hashcash — A denial of service counter-measure. 2002. http://hashcash.org/papers/hashcash.pdf.

2 Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability. In *CCS*, 2018. https://eprint.iacr.org/2018/378.

3 Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 431–448. Springer, Heidelberg, August 1999.

4 Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.

5 Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT*, 2018. http://eprint.iacr.org/2017/573.

6 Thang Dinh, Lei Fan, Jonathan Katz, Phuc Thai, and Hong-Sheng Zhou. BackPackers: A new paradigm for secure and high-performance blockchain. 2019. Manuscript. Presentation available at *Scaling Bitcoin 2019*.

7 Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 139–147. Springer, Heidelberg, August 1993.

8 Lei Fan, Jonathan Katz, and Hong-Sheng Zhou. iChing: A large-scale proof-of-stake blockchain in the open setting (or, how to mimic nakamoto's design via proof-of-stake). 2019. Manuscript at https://eprint.iacr.org/2017/656. Presentation available at *Stanford Blockchain Conference 2019*.

[6] Our design can be applied for proof-of-work based consensus e.g., Bitcoin, or proof-of-stake based consensus e.g., iChing. The solution can be a solution to proof-of-work or proof-of-stake puzzle, depending on the underlying consensus.

9 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015.

10 Jonathan Katz and Hong-Sheng Zhou. A proof-of-stake blockchain with post quantum security. 2019. Working draft.

11 Rakesh Kumar and Keith W Ross. Peer-assisted file distribution: The minimum distribution time. In *Hot Topics in Web Systems and Technologies, 2006. HOTWEB'06. 1st IEEE Workshop on*, pages 1–11. IEEE, 2006.

12 Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 597–612. Springer, Heidelberg, August 2002.

13 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. https://bitcoin.org/bitcoin.pdf.

14 Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 643–673. Springer, Heidelberg, April / May 2017.

15 Rafael Pass and Elaine Shi. FruitChains: A fair blockchain. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *36th ACM PODC*, pages 315–324. ACM, July 2017.

16 Leonid Reyzin, Dmitry Meshkov, Alexander Chepurnoy, and Sasha Ivanov. Improving authenticated dynamic dictionaries, with applications to cryptocurrencies. In Aggelos Kiayias, editor, *FC 2017*, volume 10322 of *LNCS*, pages 376–392. Springer, Heidelberg, April 2017.