OHIE: Blockchain Scaling Made Simple

Haifeng Yu Ivica Nikolić Ruomu Hou Prateek Saxena
Department of Computer Science, National University of Singapore
haifeng@comp.nus.edu.sg inikolic@nus.edu.sg houruomu@comp.nus.edu.sg prateeks@comp.nus.edu.sg

***Abstract*—Many blockchain consensus protocols have been proposed recently to *scale* the throughput of a blockchain with available bandwidth. However, these protocols are becoming increasingly complex, making it more and more difficult to produce proofs of their security guarantees. We propose a novel permissionless blockchain protocol OHIE which explicitly aims for *simplicity*. OHIE composes as many parallel instances of Bitcoin's original (and simple) backbone protocol as needed to achieve excellent throughput. We formally prove the safety and liveness properties of OHIE. We demonstrate its performance with a prototype implementation and large-scale experiments with up to $50,000$ nodes. In our experiments, OHIE achieves linear scaling with available bandwidth, providing about 4-10Mbps transaction throughput (under 8-20Mbps per-node available bandwidth configurations) and at least about 20x better decentralization over prior works.**

I. INTRODUCTION

Blockchain protocols power several open computational platforms, which allow a network of nodes to agree on the state of a distributed ledger periodically. The distributed ledger provides a total order over *transactions*. Nodes connect to each other via an open peer-to-peer (P2P) overlay network, which is *permissionless*: it allows any computer to connect and participate in the computational service without registering its identity with a central authority. A blockchain consensus protocol enables every node to *confirm* a set of transactions periodically, batched into chunks called blocks. The protocol ensures that honest nodes all agree on the same total ordering of the confirmed blocks, and that the set of confirmed blocks grow over time. The earliest such protocol was in Bitcoin [39], and has spurred interest in many blockchain platforms since.

The seminal Bitcoin protocol, published about a decade ago [39], laid some of the key foundations for modern blockchain protocols. But as Bitcoin gained popularity, its low throughput has been cited as glaring concern resulting in high costs per transaction [10]. Currently, Ethereum and Bitcoin process only about 5KB or 10 transactions per second on average, which is less than 0.2% of the average available bandwidth in their respective P2P networks [19]. Many recent research efforts have thus focused on improving the transaction throughput, resulting in a series of beautiful designs for permissionless blockchains [2], [12], [13], [20], [27], [28], [32], [33], [45], [47].

Bitcoin's core consensus protocol—called Nakamoto consensus—still stands out in one critical aspect: it is remarkably *simple*. Nakamoto consensus can be fully described in a few dozens of lines of pseudo-code. Such simplicity makes it extensively amenable to re-parameterization in hundreds of deployments, and more importantly, a series of formal proofs on its security guarantees have been carefully and independently established in several research works [17], [18], [24], [26], [40].

The importance of keeping constructions *simple* enough to allow such formal proofs and cross validations cannot be over-emphasized. Consensus protocols are notoriously difficult to analyze in the presence of byzantine failures. Formal proofs/analysis are especially important to protocols that are difficult to upgrade once deployed: Upgrades of blockchain protocols after deployment (i.e., "hard forks") cause both philosophical disagreements and financial impact.

Some recent high-throughput blockchain protocols do strive to retain the simplicity of Nakamoto consensus. Unfortunately, many of them do not come with formal end-to-end security proofs. As an example, Conflux [32] is a recent high-throughput blockchain protocol with an elegant design. But it has only provided informal security arguments (Section 3.3 in [32]). Our extended technical report [46] shows that in our simulation, as the throughput of Conflux increases, the security properties of Conflux deteriorate.[1] Such an undesirable property of Conflux is hard to discover via informal arguments. The Conflux paper itself also presented effective attacks on a prior protocol (Phantom [45]) that comes without proofs.

Our goal. This work aims to develop a *simple* blockchain consensus protocol, which should admit formal end-to-end proofs on safety and liveness, while retaining the high throughput achieved by state-of-the-art blockchain protocols. Specifically, we aim to achieve:

1) *Near-optimal resilience*: Tolerate an adversarial computational power fraction f close to $\frac{1}{2}$, which is near-optimal;
2) *Throughput approaching a significant fraction of the raw network bandwidth*: The raw available network bandwidth in the P2P network constitutes a crude throughput upper bound for all blockchain protocols. We aim to achieve a throughput, in terms of transactions processed per second, that approaches a significant fraction of this raw network bandwidth.[2]

[1] Related observations on Conflux have also been independently made by Bagaria et al. [5] and Fitzi et al. [15].

[2] Note that the raw available bandwidth is only a rather *crude* upper bound, and hence in practice it is unlikely for the throughput to reach this upper bound. For example, this crude upper bound does not take into account factors such that TCP slow start, probabilistic block generation, probabilistic hot-spots in the P2P overlay network, overheads for determining which blocks to gossip, and so on.

DOI 10.1109/SP40000.2020.00008

3) *Decentralization*: Many dynamically selected block proposers should be able to add blocks to the blockchain per second, rather than for example, having one leader or a small committee add blocks over a long period of time. More block proposers make transactions less susceptible to censorship [34], [36], and a DoS attack against a small number of nodes will no longer impact the availability of the entire system [23].

Our approach. This work proposes OHIE,[3] a novel blockchain protocol for the permissionless setting. Specifically, OHIE composes many *parallel* instances of the Nakamoto consensus protocol. OHIE first applies a simple mechanism to force the adversary to evenly split its adversarial computational power across all these chains (i.e., Nakamoto consensus instances). Next, OHIE proposes a simple solution to securely arrive at a global order for blocks across all the parallel chains, hence achieving consistency.

The *modularity* of OHIE enables us to prove (under any given constant $f < \frac{1}{2}$) its safety and liveness properties via a *reduction* from those of Nakamoto consensus. Our proof invokes existing theorems on Nakamoto consensus, making the proof modular and streamlined [40]. By running as many (e.g., 1000) chains as the network bandwidth permits, OHIE's throughput scales with available network bandwidth. Finally, the parallel chains in OHIE lead to excellent decentralization, since many miners can simultaneously add new blocks.

Our results. We have implemented a prototype of OHIE, the source code of which is publicly available [1]. We have evaluated it on Amazon EC2 with up to $50,000$ nodes, under similar settings as in prior works [20], [32]. Our evaluation first shows that OHIE's throughput scales linearly with available bandwidth, as is the case with state-of-the-art protocols [13], [20], [28], [33], [47]. For example, under configurations with 8-20Mbps per-node bandwidth, OHIE achieves about 4-10Mbps transaction throughput. This translates to close to 1000 to 2500 transactions per second, assuming 500-byte average transaction size as in Bitcoin. Such throughput is about 550% of the throughput of AlgoRand [20] and 150% of the throughput of Conflux [32] under similar available bandwidth. This suggests that while explicitly focusing on simplicity, OHIE retains the high throughput property of modern blockchain designs. Second, regardless of the throughput, the confirmation latency for blocks in OHIE is always below 10 minutes in our experiments, under security parameters comparable to Bitcoin and Ethereum deployments. (The confirmation latencies in Bitcoin and Ethereum are 60 minutes and 3 minutes, respectively.) Finally, our experiments show that the decentralization factor of OHIE is at least about 20x of all prior works.

II. SYSTEM MODEL AND PROBLEM

System model. Our system model and assumptions directly follow several prior works (e.g., [26], [40]). We model hash functions as random oracles, and assume that some random *genesis blocks* are available from an initial trusted setup. We consider a permissionless setting, where nodes have no pre-established identities. We use standard proof-of-work (PoW) puzzles, a form of sybil resistance, to limit the adversary by computation power. We assume that the entire network has total n units of computational power, and some reasonable estimation of n is known. Out of this, the *adversary* controls fn units of computational power, with f being any constant below $\frac{1}{2}$. The adversary can deviate arbitrarily from the prescribed protocol, and hence is *byzantine*. We assume that some procedure to estimate the total computation power exists a-priori [18]. Standard PoW schemes help ascertain this periodically. For instance, in Bitcoin, the rate of PoW solutions is adjusted (periodically) to be approximately 10 minutes, and the PoW difficulty essentially maps to the estimated total computation power in the network. We can use the same mechanism in our design.

Given a fixed block size (e.g., 20 KB), we assume that honest nodes form a well-connected synchronous P2P overlay network, so that an honest node can broadcast (via gossiping) such a block with a maximum latency of δ to other honest nodes. Our protocol, much like Bitcoin, can tolerate variations in the actual propagation delay. Network partitioning attacks can delay block delivery arbitrarily and can cause honest nodes to lose inter-connectivity [4]. Defences to mitigate these attacks are an important area of research; however, they are outside the scope of the design of the consensus protocol. If the network becomes completely asynchronous, blockchain consensus is considered impossible [40]. In the presence of partitions, the CAP theorem suggests that protocols can either choose liveness or safety, but not both [21], [22]; we choose liveness—the same as Nakamoto consensus. The adversary sees every message as soon as it is sent. The adversary can arbitrarily inject its own messages into the system at any time (this captures the *selfish mining* attack [14], where newly mined blocks are injected at strategic points of time).

Problem definition. A blockchain protocol should enable any node at any time to output a sequence of total-ordered blocks, which we call the *sequence of confirmed blocks* (or SCB in short). For example, in the Bitcoin protocol, the SCB is simply the blockchain itself after removing the last 6 blocks. *Safety* and *liveness*, in the context of blockchain protocols, correspond to the **consistency** and **quality-growth** properties of the SCB. Informally, these two properties mean that the SCB's on different honest nodes at different time are always consistent with each other, and that the total number of *honest blocks* (i.e., blocks generate by honest nodes) in the SCB grows over time at a healthy rate. We leave the formal definitions of these properties to Section V.

Having **consistency** and **quality-growth** is sufficient to enable a wide range of different applications. For example, **consistency** prevents double-spending in a cryptocurrency—if two transactions spend the same coin, all nodes honor only the

[3]The word "ohie" comes from the Maori language and means "simple".

first transaction in the total order [4]. Similarly, any "conflicting" state updates by smart contracts running on the blockchain can be ordered consistently by all nodes, by following the ordering in the SCB. In fact, ensuring a total order is key to support many different consistency properties [6], [29], [30] for applications, including for smart contracts [3].

Finally, if the same transaction is included in multiple blocks in the SCB, then the first occurrence will be processed, while the remaining occurrences will be ignored. To avoid such waste, miners should ideally pick different transactions to include in blocks. For example, among all transactions, a miner can pick those transactions whose hashes are the closest to the hash of the miner's public key. Note that this does not impact safety or liveness at all; it simply reduces the possibility of multiple inclusions (in different blocks) of the same transaction.

III. CONCEPTUAL DESIGN

OHIE composes k (e.g., $k = 1000$) *parallel* instances of Nakamoto consensus. Intuitively, we also call these k parallel instances as k parallel chains. Each chain has a distinct genesis block, and the chains have ids from 0 to $k-1$ (which can come from the lexicographic order of all the genesis blocks). Within each instance, we follow the longest-path-rule in Nakamoto consensus.[5] The miners in OHIE extend the k chains concurrently.

A. Mining in OHIE

Consider any fixed block size (e.g., 20 KB), and the corresponding block propagation delay δ (e.g., 2 seconds). Existing results [26], [40] on Nakamoto consensus show that for any given constant $f < \frac{1}{2}$, there exists some constant c such that if the *block interval* (i.e., average time needed to generate the next block on the chain) is at least $c \cdot \delta$, then Nakamoto consensus will offer some nice security properties. (Theorem 2 later makes this precise.) As an example, for $f = 0.43$, it suffices [40] for $c = 5$. Such $c \cdot \delta$, together with n, then maps to a certain *PoW difficulty* p in Nakamoto consensus, which is the probability of mining success for one hash operation. (Theorem 2 gives the precise mapping.) Given p, the hash of a valid block in Nakamoto should have $\log_2 \frac{1}{p}$ leading zeros.

In OHIE, to tolerate the same f as above, the hash of a valid block should have $\log_2 \frac{1}{kp}$ leading zeros, where the value p is chosen to be the same as in the above Nakamoto consensus protocol. Next, the last $\log_2 k$ bits of the hash[6] of the OHIE block will index to one of the k chains in OHIE, and the block will be assigned to and will extend from that chain. We have assumed the hash function to be a random oracle, and note that $\log_2 \frac{1}{kp} + \log_2 k = \log_2 \frac{1}{p}$. Hence for any *given* chain in OHIE, the probability of one hash operation (*either* done by honest nodes *or* done by the adversary) generating a block for that chain[7] is exactly p, which is the same as in Nakamoto consensus. Similarly, the block interval for any given chain in OHIE will be the same as in Nakamoto consensus.

The above relation can be formalized: Taking all mechanisms in OHIE (especially the Merkle tree mechanism described next) into account, Lemma 3 later will prove that the behavior of any given chain in OHIE almost follows exactly the same distribution as the behavior of the single chain in Nakamoto consensus. Note that different chains in OHIE are still *correlated*, since a block is assigned to exactly one chain. But we will be able to properly bound the probability of all bad events (whether correlated or not) via a simple union bound.

Finally, since OHIE has k parallel chains, on expectation there will be total k blocks (across all chains) generated every $c \cdot \delta$ time, instead of just one block. Our experiments later will confirm the following simple yet critical property: *Propagating many parallel blocks has minimal negative impact on the block propagation delay δ, as compared to propagating a single such block, until we start to saturate the network bandwidth of the system.* Hence in OHIE, we use as large a k as possible to effectively utilize all the bandwidth in the system, subject to the condition that δ is minimally impacted.

B. Security of Individual Chains

In Nakamoto consensus, a new block B extends from some existing block A. The PoW computes over B, which contains the hash of A as a field, cryptographically binding the extension of A by B. In OHIE, however, a miner does not know which chain a new block will extend until it finishes the PoW puzzle, the last $\log_2 k$ bits of which then determine the chain extended.

To deal with this, in OHIE, a miner uses a Merkle tree [37] to bind to the last blocks of all the k chains in its local view. Specifically, let A_i be the last block[8] of chain i, for $0 \le i \le k-1$. The miner computes a Merkle tree using $hash(A_0)$ through $hash(A_{k-1})$ as the tree leaves. The root of the Merkle tree is included in the new block B as an input to the PoW puzzle. After B is mined, the integer i that corresponds to the last $\log_2 k$ bits of $hash(B)$ determines the block A_i from which B extends. When disseminating B in the network, a miner includes $hash(A_i)$ and the Merkle proof of $hash(A_i)$ in the message. (The value of i will be directly obtained from the hash of B.) The Merkle proofs are standard, consisting of $\log_2 k$ off-path hashes [37].

Intuitively, the above design binds each successful PoW to a *single* existing block on a *single* chain from which

[4] Transactions spending the same coin as earlier ones in the total order can simply be skipped as invalid by the user.

[5] Nakamoto consensus, strictly speaking, maintains a tree of blocks [39]. The longest-path-rule selects the longest path from the root (i.e., genesis block) to some leaf of the tree, where the leaf is chosen such that the path length is maximized. This path is often referred to as the chain.

[6] Under our random oracle assumption, any $\log_2 k$ bits of the hash (other that the first $\log_2 \frac{1}{kp}$ bits) work. In implementation, one can choose to use any portion (other that the first $\log_2 \frac{1}{kp}$ bits) of the hash as appropriate, based on the specific hash function.

[7] Namely, total $\log_2 \frac{1}{p}$ positions in the block's hash must match some predetermined values, respectively.

[8] Exactly the same as in Nakamoto consensus, the last block here refers to the very last block on the longest path from the genesis block. In particular, this last block is not yet partially-confirmed. (In fact, none of the last T blocks on the path are partially-confirmed.)

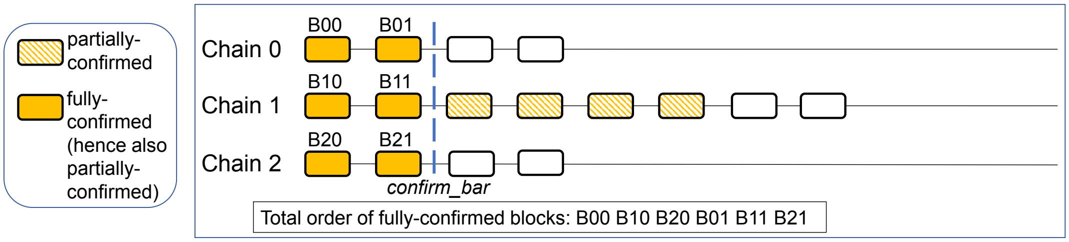


Fig. 1: Illustrating `confirm_bar` under $T = 2$. Here chain 0, 1, and 2 have 2, 6, and 2 partially-confirmed blocks, respectively. On chain 1, only the first 2 blocks are fully-confirmed.

the new block extends. For further understanding, let us consider the following example scenario. The adversary may intentionally choose A_0 through A_{k-1} all from (say) chain 3, for constructing the Merkle tree. Assume the adversary finds a block B whose hash has $\log_2 \frac{1}{kp}$ leading zeros. If the last $\log_2 k$ bits of B does not equal to 3, then B will not be accepted by any honest node. Otherwise B will be accepted, and B can only extend from A_3 (instead of any other block) on chain 3. The reason is that the honest nodes will need to verify the 4th leaf (which corresponds to chain 3) on the Merkle tree. Only A_3 can pass such verification. Also note that the adversary may intentionally not use the last block on chain 3 as A_3. This is not a problem, since the security of OHIE ultimately inherits (see Section V) from the security of Nakamoto consensus [40], and the adversary in Nakamoto consensus can already extend from any block (instead of extending from the last block).

C. Ordering Blocks across Chains – A Starting Point

Section V will show that each individual chain in OHIE inherits the proven security properties of Nakamoto consensus [40]. For example, with high probability, all blocks on a chain except the last T blocks (for some parameter T) are *confirmed* — the ordering of these confirmed blocks on the chain will no longer change in the future. This however does not yet give us a total ordering of all the confirmed blocks across all the k chains in OHIE. Recall from Section II that a node needs to generate an SCB (i.e., a total order of all confirmed blocks) satisfying **consistency** and **quality-growth**. To avoid notational collision, from this point on, we call all blocks on a chain except the last T blocks as *partially-confirmed*. Once a partially-confirmed block is added to SCB, it becomes *fully-confirmed*.

One way to design the SCB is to first include the first partially-confirmed block on each of the k chains (there are total k such blocks, and we order them by their chain ids), and then add the second partially-confirmed block on each of the k chains, and so on. This would work well, if every chain has the same number of partially-confirmed blocks.

When the chains do not have the same number of partially-confirmed blocks, we will need to impose a *confirmation bar* (denoted as `confirm_bar`) that is limited by the chain with the smallest number of partially-confirmed blocks (see Figure 1). Blocks after `confirm_bar` cannot be included in the total order yet. This causes a serious problem, since with blocks extending chains at random, some chains can have more blocks than others. In fact in our experiments (results not shown), such imbalance appears to even grow unbounded over time.

D. Ordering Blocks across Chains – Our Approach

Imagine that the longest chain is 8 blocks longer than the shortest chain. Our basic idea to overcome the previous problem is that when the next block on the shortest chain is generated, we simply view it as 8 blocks worth. Figure 2 illustrates this idea. Here each block has two additional fields used for ordering blocks across chains, denoted as a tuple (`rank`, `next_rank`). In the total ordering of fully-confirmed blocks, the blocks are ordered by increasing `rank` values, with tie-breaking based on the *chain ids*. The *chain id* of a block is simply the id of the chain to which the block belongs. For any new block B that extends from some existing block A, we directly set B's `rank` to be the same as A's `next_rank`. Putting it another way, A's `next_rank` specifies (and fixes) the `rank` of B. A genesis block always has `rank` of 0 and `next_rank` of 1.

Determining `next_rank`. Properly setting the `next_rank` of a new block B is key to our design. A miner sees all the chains, and can infer the expected `rank` of the next upcoming block on each chain (before B is added to its chain). For example, at time t_1 in Figure 2, the `next_rank` of the current last block (not the last *partially-confirmed* block) on each of the three chains is 1, 5, and 1, respectively. Hence these will be the `rank` of the upcoming blocks on those chain, respectively. Let x denote the maximum (i.e., 5) among these values, and x corresponds to the "longest" chain (in terms of rank) among the k chains. Regardless of which chain the new block B ends up belonging to, we want B to help that chain to increase its rank to catch up with the "longest" chain. Hence the node generating B should directly set B's `next_rank` to be x, or any value larger than[9] x. (To prevent adversarial manipulation, a careful implementation requiring an additional `trailing`

[9]Using a value larger than x will cause B's chain to exceed the length of the currently "longest" chain (in terms of rank). This is not a problem since the other chains will catch up with B's chain once a new honest block is added to each of those chains. We do not need the chains to have exactly the same length all the time.

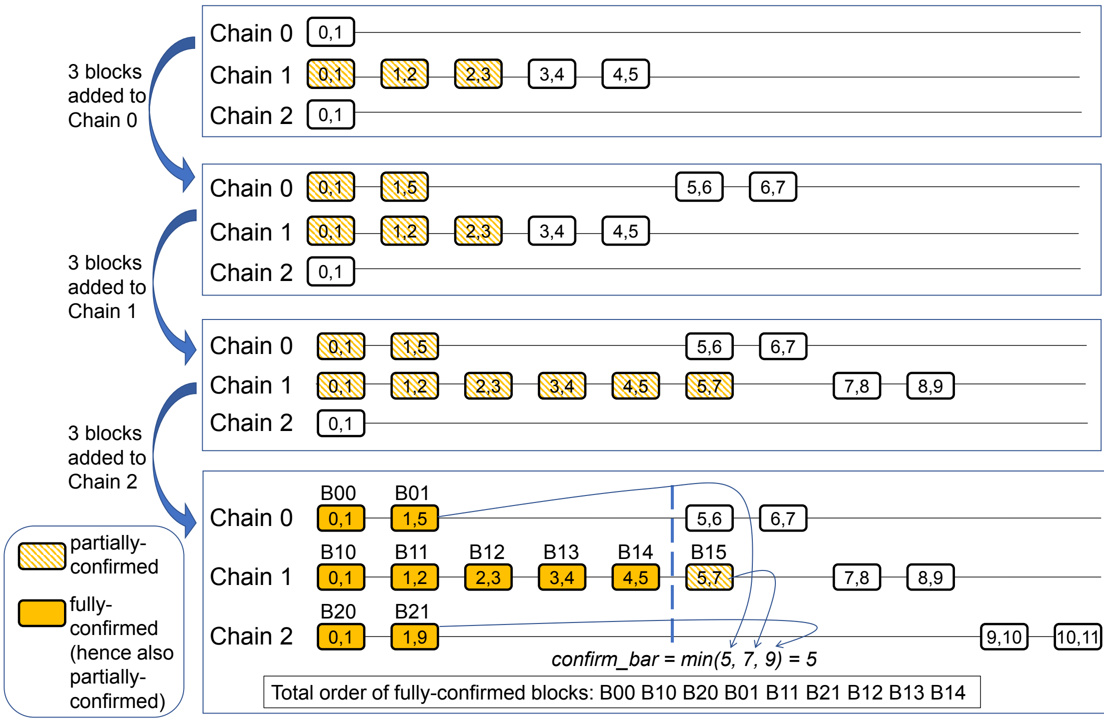


Fig. 2: Each block has a tuple (`rank`, `next_rank`). Our example here assumes that once a block is generated, it is seen by all nodes immediately. Note that OHIE does not need such an assumption. We use $T = 2$ in this figure.

field is needed— see Section IV.) Finally, we always ensure that B's `next_rank` is at least one larger than B's `rank`, regardless of x. This guarantees that the `rank` values of blocks on one chain are always increasing.

In the example in Figure 2, from time t_1 to t_2, there are 3 new blocks (with tuples $(1, 5)$, $(5, 6)$, and $(6, 7)$, respectively) added to chain 0. For the first new block added (i.e., the block with tuple $(1, 5)$), the value of x is 5, and hence the block's `next_rank` is set to be 5. For the second new block (i.e., the block with tuple $(5, 6)$), the value of x is still 5, while the `rank` of this block is already 5. Hence we set the block's `next_rank` to 6.

Determining the total order. We can now establish a total order among the blocks in the following way. Consider any given honest node at any given time and its local view of all the chains. Let y_i be the `next_rank` of the last partially-confirmed block on chain i in this view. For example, at time t_4 in Figure 2, we have $y_0 = 5$, $y_1 = 7$ and $y_2 = 9$. Note that the position of a partially-confirmed block on its respective chain will not change anymore, and hence all these y_i's are "stable". Let `confirm_bar` $\leftarrow \min_{i=1}^{k} y_i$. Then, the next partially-confirmed block on any chain must have a `rank` no smaller than `confirm_bar`. This means that the node must have seen all partially-confirmed blocks whose `rank` is smaller than `confirm_bar`. Thus, it is safe (see Lemma 4) to deem all partially-confirmed blocks whose rank is smaller than `confirm_bar` as fully-confirmed, and include them in SCB. Finally, all the fully-confirmed blocks will be ordered by increasing `rank` values, with tie-breaking favoring smaller chain ids. As an example, in Figure 2, at time t_4, we have `confirm_bar` being 5. Hence, the 9 partially-confirmed blocks whose `rank` is below 5 become fully-confirmed.

Summary. By properly setting the `rank` values of all the blocks, we ensure that the chains remain balanced in terms of the `rank`'s of their respective last blocks. This is regardless of how imbalanced the chains are in terms of the total number of blocks, hence avoiding the earlier imbalance problem in Section III-C.

IV. IMPLEMENTATION DETAILS

We call the Nakamoto consensus protocol (p, λ, T)-*Nakamoto*, where the hash of a valid block needs to have $\log_2 \frac{1}{p}$ leading zeros, λ is the security parameter (i.e., the length of the hash output), and T is the number of blocks that we remove from the end of the chain in order to obtain partially-confirmed blocks. We call the OHIE protocol (k, p, λ, T)-OHIE, where λ and T are the same as above, and where k is the number of chains in OHIE. In (k, p, λ, T)-OHIE, the hash of a valid block should have $\log_2 \frac{1}{kp}$ leading zeros.

For simplicity, the value of k is fixed in our current design of OHIE—adjusting k on-the-fly could potentially be possible via a view change mechanism, but we consider it as future work. The value of T in OHIE can be readily adjusted on-the-fly by individual nodes, without needing any coordination. Because the security of OHIE inherits from the security of Nakamoto

1: $d \leftarrow \log_2(\frac{1}{kp})$;
2: $V_i \leftarrow \{$(genesis block of chain i, the attachment for genesis block of chain i)$\}$, for $0 \le i \le k-1$;
3: $M \leftarrow$ Merkle tree of the hashes of the k genesis blocks;
4: `trailing` $\leftarrow$ hash of genesis block of chain 0;
5:
6: OHIE() {
7: **repeat forever** {
8: ReceiveState();
9: Mining();
10: SendState();
11: }
12: }
13:
14: Mining() {
15: B.`transactions` $\leftarrow$ get_transactions();
16: B.`root` $\leftarrow$ root of Merkle tree M;
17: B.`trailing` $\leftarrow$ `trailing`;
18: B.`nonce` $\leftarrow$ new_nonce();
19: $\widehat{B}$.`hash` $\leftarrow hash(B)$;
20: **if** ($\widehat{B}$.`hash` has d leading zeroes) {
21: $i \leftarrow$ last $\log_2 k$ bits of $\widehat{B}$.`hash`;
22: $\widehat{B}$.`leaf` $\leftarrow$ leaf i of M;
23: $\widehat{B}$.`leaf_proof` $\leftarrow M$.MerkleProof(i);
24: ProcessBlock(B, $\widehat{B}$);
25: }
26: }
27:
28: SendState() {
29: send V_i ($0 \le i \le k-1$) to other nodes;
30: // In implementation, only need to send those blocks not sent before.
31: }
32:
33: ReceiveState() {
34: **foreach** $(B, \widehat{B}) \in$ received state **do**
35: ProcessBlock(B, $\widehat{B}$);
36: // A block B_1 should be processed before B_2, if $\widehat{B_2}$.`leaf` or $\widehat{B_2}$.`trailing` points to B_1.
37: }

38: ProcessBlock(B, $\widehat{B}$) {
39: // do some verifications
40: $i \leftarrow$ last $\log_2 k$ bits of $\widehat{B}$.`hash`;
41: verify that $\widehat{B}$.`hash` has d leading zeroes;
42: verify that $hash(B) = \widehat{B}$.`hash`;
43: verify that $\widehat{B}$.`leaf` is leaf i in the Merkle tree, based on B.`root` and $\widehat{B}$.`leaf_proof`;
44: verify that $\widehat{B}$.`leaf` $= \widehat{A}$.`hash` for some block $(A, \widehat{A}) \in V_i$;
45: verify that B.`trailing` $= \widehat{C}$.`hash` for some block $(C, \widehat{C}) \in \cup_{j=0}^{k-1} V_j$;
46: **if** (any of the above 5 verifications fail) **then return**;
47:
48: // compute `rank` and `next_rank` values
49: $\widehat{B}$.`rank` $\leftarrow \widehat{A}$.`next_rank`;
50: $\widehat{B}$.`next_rank` $\leftarrow \widehat{C}$.`next_rank`;
51: **if** ($\widehat{B}$.`next_rank` $\le \widehat{B}$.`rank`) **then** $\widehat{B}$.`next_rank` $\leftarrow \widehat{B}$.`rank` $+ 1$;
52:
53: // update local data structures
54: $V_i \leftarrow V_i \cup \{(B, \widehat{B})\}$;
55: update `trailing`;
56: update Merkle tree M;
57: }
58:
59: OutputSCB() {
60: // determine partially-confirmed blocks and `confirm_bar`
61: **for** ($i = 0$; $i < k$; i++) {
62: $W_i \leftarrow$ get_longest_path(V_i);
63: $\texttt{partial}_i \leftarrow$ blocks in W_i except the last T blocks;
64: $(B_i, \widehat{B_i}) \leftarrow$ the last block in $\texttt{partial}_i$;
65: $y_i \leftarrow \widehat{B_i}$.`next_rank`;
66: }
67: `all_partial` $\leftarrow \cup_{i=0}^{k-1} \texttt{partial}_i$;
68: `confirm_bar` $\leftarrow \min_{i=0}^{k-1} y_i$;
69:
70: // determine fully-confirmed blocks and SCB
71: $L \leftarrow \emptyset$;
72: **foreach** $(B, \widehat{B}) \in$ `all_partial` {
73: **if** ($\widehat{B}$.`rank` $<$ `confirm_bar`) **then** $L \leftarrow L \cup \{(B, \widehat{B})\}$;
74: }
75: sort blocks in L by `rank`, tie-breaking favoring smaller chain id;
76: **return** L;
77: }

Fig. 3: Pseudo-code of the (k, p, λ, T)-OHIE protocol.

consensus, using different T values in OHIE has a similar effect as in Nakamoto consensus. For example, a user can use a larger T for a higher security level. To simplify discussion, however, we consider some fixed T value.

Overview. Figure 3 gives the pseudo-code of OHIE, as run by each node. In the main loop (Line 8 to 10) of OHIE, a node receives messages from others, makes one attempt for solving the PoW (i.e., makes one query to the random oracle or hash function), and then sends out messages. The messages contain OHIE blocks. Such a main loop is exactly the same as in Nakamoto consensus [26], [40]. Note that the block generation rate is significantly lower than the rate of queries to the random oracle. Hence, most of the time, the main loop will not have any new messages to receive/send. In an actual implementation, such sending/receiving of messages can be done in a separate thread.

The function OutputSCB() can be invoked whenever needed. It produces the current SCB, by exactly following the description in Section III-D.

Key data structures. Each OHIE node maintains sets V_0 through V_{k-1}. V_i initially contains the genesis block for chain i. During the execution, V_i is a tree of blocks, containing all those blocks with a path to that genesis block. We use W_i to denote the longest path (from the genesis block to some leaf) on this tree. All blocks on W_i, except the last T blocks, are *partially-confirmed*.

The Merkle tree M is constructed by using the hash of the very last block on W_i ($0 \le i \le k-1$) as the k leaves. Each node further maintains a `trailing` variable, which is the hash of the *trailing block*. The *trailing block* is the block with the largest next_rank value, among all blocks in $\cup_{i=0}^{k-1} V_i$. (Note that the trailing block may or may not be in $\cup_{i=0}^{k-1} W_i$.) If there are multiple such blocks, we let the trailing block be the one with the smallest chain id. Both M and trailing should be properly updated whenever the node receives new blocks.

Block generation/verification. A block B in OHIE consists of some transactions, a fresh nonce, the Merkle root, and a B.trailing field. All these fields are fed into the hash function, during mining. Section III-B explains why we include the Merkle root. The following explains the B.trailing field.

As explained in Section III-D, regardless of which chain the new block B ends up belonging to, we want B to help that chain to increase its rank to catch up with the "longest chain" (in terms of rank). To do so, all we need is to set B's next_rank to be large enough. A naive design is to let the creator of B directly set B's next_rank, based on its local V_i's. Such a design enables the adversary to pick the maximum possible value[10] for that field. Doing so exhausts the possible values for next_rank, since the next_rank of blocks on a given chain needs to keep increasing.

This is why in OHIE, each node maintains a trailing variable (i.e., the hash of the trailing block[11]). The miner sets B.trailing $\leftarrow$ trailing, and B.trailing (as part of B) is fed into the hash function. When a node u receives a new block B, the node will verify whether it has already seen some block C whose hash equals B.trailing. If so, u sets next_rank of B to be the same as the next_rank of C. Otherwise B will not be accepted. Doing so prevents the earlier attack.

Note that the adversary can still lie, and claim some arbitrary block as its trailing block. Theorem 1 in Section V, however, will prove that this does *not* cause any problem. Intuitively, by doing so, the adversary is simply refusing to help a chain to grow its rank to catch up. But the next honest block on the chain will enable the chain to grow its rank properly, and to immediately catch up.

Block attachment. Some information about a block is not available until after the block is successfully mined, and we include such information in an *attachment* for the block. The attachment for a block is always stored/disseminated together with the block. We always use $\widehat{B}$ to denote an attachment for a block B. (Note that $\widehat{B}$ is not fed into the hash function when we compute the hash of B.) Specifically, let i be the last $\log_2 k$ bits in B's hash. $\widehat{B}$ will include leaf i of the Merkle tree, and the corresponding Merkle proof. For convenience, $\widehat{B}$ also contains the hash of B, in its $\widehat{B}$.hash field.

An attachment $\widehat{B}$ further contains a rank value and a next_rank value, for the block B. (The block B itself actually does not have a rank or next_rank field.) When a node receives a new block B and its attachment $\widehat{B}$, the node independently computes the proper values for $\widehat{B}$.rank and $\widehat{B}$.next_rank based on its local information, and use those values (Line 49 to 51 in Figure 3). Lemma 5 will prove that except with an exponentially small probability (i.e., excluding hash collisions and so on), for any block B, all honest nodes will assign *exactly the same value* to $\widehat{B}$.rank ($\widehat{B}$.next_rank). Finally, a genesis block B always has $\widehat{B}$.rank $= 0$ and $\widehat{B}$.next_rank $= 1$.

[10]next_rank must have a finite domain in actual implementation.

[11]Our trailing block is the block with the largest next_rank in $\cup_{i=0}^{k-1} V_i$. One could further require the trailing block to be in $\cup_{i=0}^{k-1} W_i$. The security guarantees of OHIE (in Section V) and our proofs also hold for such an alternative design.

V. Security Guarantees of OHIE

Our analysis results presented in this section hold under *all possible strategies* of the adversary.

A. Overview of Guarantees

Formal framework. Our formal framework directly follows several prior works (e.g., [26], [40]). All executions we consider are of polynomial length with respect to the security parameter λ. We model hash functions as random oracles. The execution of the system comprises a sequence of *ticks*, where a *tick* is the amount of time needed to do a single proof-of-work query to the random oracle by an honest node. Hence in each tick, each honest node does one such query, while the adversary does up to fn such queries. We allow these fn queries to be done sequentially, which only makes our results stronger. Define Δ (e.g., 2×10^{12}) to be δ (e.g., 2 seconds) divided by the duration of a tick (e.g., 10^{-12} second) — hence a message sent by an honest node will be received by all other honest nodes within Δ ticks. A block is an *honest block* if it is generated by some honest node, otherwise it is a *malicious block*. For two sequences S_1 and S_2, S_2 is a *prefix* of S_1 iff S_1 is the concatenation of S_2 and some sequence S_3. Here S_3 may be empty — hence S_1 is also a prefix of itself. Finally, recall from Section IV the definitions of (p, λ, T)-Nakamoto and (k, p, λ, T)-OHIE.

Main theorem on OHIE. We will eventually prove the following:

Theorem 1. *Consider any given constant $f < \frac{1}{2}$. Then there exists some positive constant c such that for all $p \le \frac{1}{c\Delta n}$ and all $k \ge 1$, the (k, p, λ, T)-OHIE protocol satisfies all the following properties, with probability at least $1 - k \cdot exp(-\Omega(\lambda)) - k \cdot exp(-\Omega(T))$:*

- **(growth)** *On any honest node, the length of each of the k chains increases by at least T blocks every $\frac{2T}{pn}$ ticks.*
- **(quality)** *On any honest node and at any time, every T consecutive blocks on any of the k chains contain at least $\frac{1-2f}{1-f}T$ honest blocks.*
- **(consistency)** *Consider the SCB S_1 on any node u_1 at any time t_1, and the SCB S_2 on any node u_2 at any time t_2.*[12] *Then either S_1 is a prefix of S_2 or S_2 is a prefix of S_1. Furthermore, if ($u_1 = u_2$ and $t_1 < t_2$) or ($u_1 \ne u_2$ and $t_1 + \Delta < t_2$), then S_1 is a prefix of S_2.*
- **(quality-growth)** *For all integer $\gamma \ge 1$, the following property holds after the very first $\frac{2T}{pn}$ ticks of the execution: On any honest node, in every $(\gamma+2) \cdot \frac{2T}{pn} + 2\Delta$ ticks,*

[12]Here u_1 (t_1) may or may not equal u_2 (t_2).

at least $\gamma \cdot k \cdot \frac{1-2f}{1-f} T$ honest blocks are newly added to SCB.

Values of c, λ, and T. The value of c in Theorem 1 will be *exactly the same* as the c in Theorem 2 next. If we want the properties in Theorem 1 to hold with probability $1-\epsilon$, then both λ and T should be $\Theta(\log \frac{1}{\epsilon} + \log k)$. The value of ϵ needed by a real application (e.g., a cryptocurrency system) is typically orders of magnitude smaller than $\frac{1}{k}$ — hence λ and T are usually just $\Theta(\log \frac{1}{\epsilon})$.

Four properties. The **growth** and **quality** in Theorem 1 are about the individual component chains in OHIE. For **consistency**, considering individual chains obviously is not sufficient. Hence, Theorem 1 proves **consistency**[13] for the SCB, which is the final total order of the fully-confirmed blocks. Theorem 1 also proves the **quality-growth** of SCB, showing that SCB will incorporate more honest blocks at a certain rate. Ultimately, **consistency** corresponds to the *safety* of OHIE, while **quality-growth** captures the *liveness* of OHIE.

B. Existing Result as a Building Block

Our proof later will invoke the following theorem from [40] on Nakamoto consensus.

Theorem 2. (Adapted from Corollary 3 in [40].) *Consider any given constant $f < \frac{1}{2}$. Then there exists some positive constant c such that for all $p \le \frac{1}{c\Delta n}$, the (p, λ, T)-Nakamoto protocol satisfies all the following properties, with probability at least $1 - exp(-\Omega(\lambda)) - exp(-\Omega(T))$:*

- **(growth)** *On any honest node, the length of the chain increases by at least T blocks every $\frac{2T}{pn}$ ticks.*
- **(quality)** *On any honest node and at any time, every T consecutive blocks on the chain contain at least $\frac{1-2f}{1-f} T$ honest blocks.*
- **(consistency)** *Let S_1 (S_2) be the sequence of blocks on the chain on any node u_1 (u_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.*

Combing the **growth** and **quality** properties immediately leads to **quality-growth** for (p, λ, T)-Nakamoto:

- **(quality-growth)** Let S be the sequence of blocks on the chain on any given honest node, excluding the last T blocks. Then after the very first $\frac{2T}{pn}$ ticks of the execution, in every $\frac{2T}{pn}$ ticks, at least $\frac{1-2f}{1-f} T$ honest blocks are newly added to S.

C. Proof for Theorem 1

Overview of proof for Theorem 1. Our first step is to obtain a *reduction* from (p, λ, T)-Nakamoto to (k, p, λ, T)-OHIE. Consider any given adversary $\mathcal{A}$ for (k, p, λ, T)-OHIE, and any given chain i ($0 \le i \le k-1$) in the execution of (k, p, λ, T)-OHIE against $\mathcal{A}$. Our reduction step will show that there exists some adversary $\mathcal{A}'$ for (p, λ, T)-Nakamoto with the following property: Except some exponentially small probability and after proper mapping from blocks in (k, p, λ, T)-OHIE to blocks in (p, λ, T)-Nakamoto, the behavior of chain i in the execution of (k, p, λ, T)-OHIE against $\mathcal{A}$ follows *exactly the same distribution* as the behavior of the (single) chain in the execution of (p, λ, T)-Nakamoto against $\mathcal{A}'$. Such a reduction implies that existing properties on (p, λ, T)-Nakamoto directly carry over to each individual chain in (k, p, λ, T)-OHIE.

Our second step is to show that conditioned upon all the existing properties (in Theorem 2) on (p, λ, T)-Nakamoto holding for each individual chain in (k, p, λ, T)-OHIE, the SCB generated in OHIE must satisfy the properties in Theorem 1, except with some exponentially small probability. The reasoning in this step will center around the `rank` and `next_rank` values of the blocks.

Formal concepts needed for reduction. Consider any (black-box and potentially randomized) adversary $\mathcal{A}$ for OHIE. Define $\texttt{EXEC}(\text{OHIE}, \texttt{k}, \texttt{p}, \lambda, \texttt{T}, \mathcal{A})$ to be the random variable denoting the joint states of all the honest nodes and the adversary throughout (i.e., at every tick) the entire execution resulted from running (k, p, λ, T)-OHIE against $\mathcal{A}$. Define random variable $\texttt{Chainview}_i(\texttt{EXEC}(\text{OHIE}, \texttt{k}, \texttt{p}, \lambda, \texttt{T}, \mathcal{A}))$ to be the joint state of chain i on every honest node throughout this execution. Recall that in OHIE, a node maintains k sets of blocks, V_0 through V_{k-1}, and chain i corresponds to the longest path in V_i. One could imagine that for each (u, t) pair, $\texttt{Chainview}_i()$ contains a component describing chain i on the honest node u at tick t.

We similarly define $\texttt{EXEC}(\text{Nakamoto}, \texttt{p}, \lambda, \texttt{T}, \mathcal{A}')$ for the execution resulted from running (p, λ, T)-Nakamoto against adversary $\mathcal{A}'$. Also, we similarly define random variable $\texttt{Chainview}(\texttt{EXEC}(\text{Nakamoto}, \texttt{p}, \lambda, \texttt{T}, \mathcal{A}'))$ to be the joint state of the (single) chain on every honest node throughout this execution.

For two random variables X and Y over some finite domain Z, define their *variation distance* to be $||X - Y|| = 0.5 \sum_{z \in Z} |\Pr[X = z] - \Pr[Y = z]|$. If X and Y are both parameterized by λ, we say that $X(\lambda)$ is *strongly statistically close* to $Y(\lambda)$ iff $||X - Y|| = exp(-\Omega(\lambda))$.

Our reduction lemma. The following is our reduction lemma:

Lemma 3. *Consider any given i where $0 \le i \le k-1$, and any given adversary $\mathcal{A}$ for (k, p, λ, T)-OHIE. There exists some adversary $\mathcal{A}'_i$ for (p, λ, T)-Nakamoto such that the following two random variables are strongly statistically close:*

- $\texttt{Chainview}_i(\texttt{EXEC}(\text{OHIE}, \texttt{k}, \texttt{p}, \lambda, \texttt{T}, \mathcal{A}))$
- $\sigma_i^\tau(\texttt{Chainview}(\texttt{EXEC}(\textit{Nakamoto}, \texttt{p}, \lambda, \texttt{T}, \mathcal{A}'_\texttt{i})))$

Here τ is the randomness in $\texttt{EXEC}(\textit{Nakamoto}, \texttt{p}, \lambda, \texttt{T}, \mathcal{A}'_\texttt{i})$. The random variable $\sigma_i^\tau(\texttt{Chainview}())$ is the same as $\texttt{Chainview}()$, except that we replace each block B' in $\texttt{Chainview}()$ with another block $\sigma_i^\tau(B')$. The mapping $\sigma_i^\tau()$ is some one-to-one mapping from each block B' among all the blocks on all the nodes in (p, λ, T)-Nakamoto to some block $\sigma_i^\tau(B')$ on all the nodes in (k, p, λ, T)-OHIE. The mapping $\sigma_i^\tau()$ guarantees that i) B' is an honest block iff $\sigma_i^\tau(B')$ is an

[13]Different prior works [17], [40] define consistency slightly differently. Our definition here is either equivalent or stronger than those in the prior works.

honest block, and ii) B' extends from A' iff $\sigma_i^\tau(B')$ extends from $\sigma_i^\tau(A')$.

The crux of proving this lemma is to construct the adversary $\mathcal{A}'_i$. In our proof, $\mathcal{A}'_i$ simulates a certain execution of OHIE against $\mathcal{A}$. More precisely, $\mathcal{A}'_i$ simulates all the OHIE nodes, as well as the adversary $\mathcal{A}$ in a black-box fashion. The adversary $\mathcal{A}'_i$ simultaneously interacts with the (real) nodes running (p, λ, T)-Nakamoto, while ensuring that the simulated OHIE execution and the real Nakamoto execution are properly "coupled". Due to space limitations, we defer the complete proof of this lemma to our extended technical report [46]. To be fully rigorous, the complete proof needs additional formalism and also needs to fully specify the (p, λ, T)-Nakamoto protocol by pseudo-code.

From individual chains to SCB. The following lemma (proof in Appendix A) establishes the connection from the individual chains in OHIE to the SCB in OHIE:

Lemma 4. *If the three properties in Theorem 2 hold for each of the k chains in (k, p, λ, T)-OHIE, then with probability at least $1 - exp(-\Omega(\lambda))$, the SCB in OHIE satisfies the* **consistency** *and* **quality-growth** *properties in Theorem 1.*

Final proof. Using all the lemmas, we can now prove Theorem 1:

Proof. (for Theorem 1) We set the constant c in Theorem 1 to be the same as the c in Theorem 2. For any given i where $0 \leq i \leq k-1$, Lemma 3 and Theorem 2 tell us that for chain i in (k, p, λ, T)-OHIE, with probability at least $1 - exp(-\Omega(\lambda)) - exp(-\Omega(T)) - exp(-\Omega(\lambda))$, the three properties in Theorem 2 hold for that chain. Hence with probability at least $1 - k \cdot exp(-\Omega(\lambda)) - k \cdot exp(-\Omega(T))$, the properties in Theorem 2 hold for *all* k chains in (k, p, λ, T)-OHIE. The **growth** and **quality** properties in Theorem 1 then directly follow. Applying Lemma 4 further leads to the **consistency** and **quality-growth** properties in Theorem 1. □

D. Discussion and Comparison

Plug-in alternative results. Our analysis invokes the results in [40]. The analysis in [40] is just one of the many works [17], [18], [24], [26], [40] that analyze Nakamoto-style protocols. A highlight of our proof on OHIE is that it invokes the existing guarantees on (p, λ, T)-Nakamoto as *black-box*. Hence alternative results on (p, λ, T)-Nakamoto directly translates to alternative results on OHIE.

Specifically, Theorem 2 (adopted from [40]) has the following three *quantitative measures*:

- The value of c, where $\frac{1}{c\Delta n}$ is the upper limit on p.
- The value of $x = \frac{2T}{pn}$ ticks (i.e., the *growth rate*), which is the time needed for the chain length to grow by T blocks.
- The value of $y = \frac{1-2f}{1-f}T$ (i.e., the *quality rate*), which is the number of honest blocks among every T consecutive blocks on the chain.

Other analyses [17], [18], [24], [26] have obtained alternative results on x and y values for (p, λ, T)-Nakamoto. They have also derived various sufficient conditions[14] for consistency in (p, λ, T)-Nakamoto, which all ultimately translate to requirement on the value of c.

We can directly plug in alternative c, x, and y values from alternative analyses on (p, λ, T)-Nakamoto to obtain alternative results on OHIE. If we do so, then the value of c in Theorem 1 will be *exactly the same* as the given c. The **consistency** property in Theorem 1 will remain unchanged. The remaining properties in Theorem 1 will become:

- **(growth)** The length of each of the k chains on each honest node will increase by at least T blocks every x ticks.
- **(quality)** On any honest node and at any time, every T consecutive blocks on any of the k chains must contain at least y honest blocks.
- **(quality-growth)** For all integer $\gamma \geq 1$, after the very first x ticks of the execution, on any honest node in every $(\gamma + 2) \cdot x + 2\Delta$ ticks, at least $\gamma \cdot k \cdot y$ honest blocks are newly added to SCB.

Comparing with prior results. With the above discussion, we can now easily compare Theorem 1 with any of the prior results [17], [18], [24], [26], [40]. Consider the result from any such prior analyses, with certain resulting c, x, and y values. (This also implies that in that particular result, the rate of **quality-growth** is y new honest blocks every x ticks.) Now let us plug in such c, x, and y values into Theorem 1. Then OHIE will provide exactly the same quantitative guarantees as that prior result, in terms of the value of c, the growth rate, and the quality rate. The only difference will be regarding the **quality-growth** of SCB. Under the given x and y, for all integer $\gamma \geq 1$, in OHIE $\gamma \cdot k \cdot y$ honest blocks are newly added to the SCB every $(\gamma+2) \cdot x + 2\Delta$ ticks. Since 2Δ is dominated by $(\gamma+2) \cdot x$, the average rate of honest blocks being added to the SCB is about $k \cdot y$ honest blocks every x ticks, in the long term. Such a rate is k times of the rate of **quality-growth** in the corresponding prior result. This also intuitively explains why OHIE increases throughput by k times.[15]

Confirmation latency. Finally, Theorem 1 also indirectly gives OHIE's guarantee on transaction *confirmation latency* (also called *wait-time* in [17], [40]): Assume that we want the properties in Theorem 1 to hold with $1 - \epsilon$ probability, and let us invoke the theorem with $p = \Theta(\frac{1}{c\Delta n})$. By **quality-growth**, at least one honest block is newly added to SCB within $\Theta(T\Delta) = \Theta((\log \frac{1}{\epsilon} + \log k)\Delta)$ ticks. Now if a transaction is injected into all honest nodes continuously for $\Theta((\log \frac{1}{\epsilon} + \log k)\Delta)$ ticks, this transaction must be included in the SCB after those ticks. Hence, the confirmation latency is simply $\Theta((\log \frac{1}{\epsilon} + \log k)\Delta)$ ticks. As explained earlier,

[14]For example, the sufficient condition derived in [40] is that $\alpha(1 - 2(\Delta + 1)\alpha) \geq (1 + \eta)\beta$ for some positive constant η.

[15]Of course, k cannot be unbounded. In practice, increasing k beyond a certain point will cause a non-trivial increase in Δ, when the system starts to saturate the network bandwidth. Our experiments later will quantify this.

this usually becomes $\Theta(\Delta \log \frac{1}{\epsilon})$ in practical settings. Such a confirmation latency is the same as in [17], [40], and is fundamental in all Nakamoto-style protocols: Each new block takes $\Theta(\Delta)$ ticks, and $\Theta(\log \frac{1}{\epsilon})$ new blocks are needed for the "confidence" to reach $1 - \epsilon$.

VI. Experimental Evaluation

Methodology. We have implemented a prototype of OHIE in C++, with total around 4,700 lines of code. We use Amazon's EC2 virtual machines (or EC2 instances) for evaluation. We rent `m4.2xlarge` instances in 14 cities around the globe[16], with each instance having 8 cores and 1Gbps bandwidth. The one-way latency between two random EC2 instances is about 90-140ms, which is consistent with AlgoRand's experiments [20] and with measurements in Bitcoin and Ethereum [19]. To avoid excessive monetary expenses on EC2, we run two sets of experiments. Our *macro experiments* run 12,000 to 50,000 OHIE nodes on up to 1,000 EC2 instances to evaluate the end performance of OHIE, while our *micro experiments* run 1,000 OHIE nodes on 20 EC2 instances to determine OHIE internal parameters.

In all experiments, the OHIE nodes form a P2P overlay by each node connecting to 8 randomly selected peers, and per-node bandwidth is up to 20Mbps, since we run 50 nodes per EC2 instance. This setup is the same as in the experiments of AlgoRand [20] and Conflux [32]. All results reported are averaged over 5 runs, each lasting until measurements stabilize in that run.

A. Choosing Block Size and Block Interval in OHIE

We first use micro experiments to measure the *block propagation delay* (*BPD*) for a single block (with no parallel propagations) to reach 99% of the nodes. We consider different bandwidth configurations where the per-node bandwidth ranges from 8Mbps to 20Mbps. We observe that the BPD for 20 KB blocks is about 1.7-1.9 seconds, across all our bandwidth configurations. Similar BPD values are observed for block sizes ranging from 10 KB to 64 KB.

We further observe that such BPD does not significantly increase as the network size increases: Even in our macro experiments with 50,000 nodes, the BPD values are still only about 3.2 seconds for 10-20 KB blocks. This is consistent with theoretical expectations about random graphs,[17] and the fact that BPD is proportional to the average number of hops between two nodes.

Smaller blocks enable better decentralization, but also incur more protocol overheads. Taking all factors into account, we choose a block size of 20 KB for OHIE. We then set p to correspond to a block interval of about 10 seconds on *each* chain. Based on Section V and results in [40], a block interval of $5\times$ BPD is sufficient to tolerate $f = 0.43$.

[16] This is the maximal number of cities with such instances.

[17] In random Erdos-Renyi graphs, roughly speaking, the average number of hops between two nodes increases only logarithmically with the number of nodes.

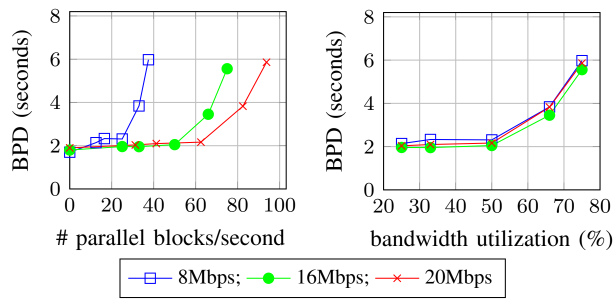


Fig. 4: BDP under different per-node bandwidth configurations (8-20Mbps). [**left**]: BPD vs. number of blocks propagated in parallel per second. The BPD for a single block (non-parallel case) is plotted as x-axis value of 0. [**right**]: BPD vs. fraction of raw bandwidth utilized by parallel block propagation.

B. Efficient Parallel Propagation of Blocks

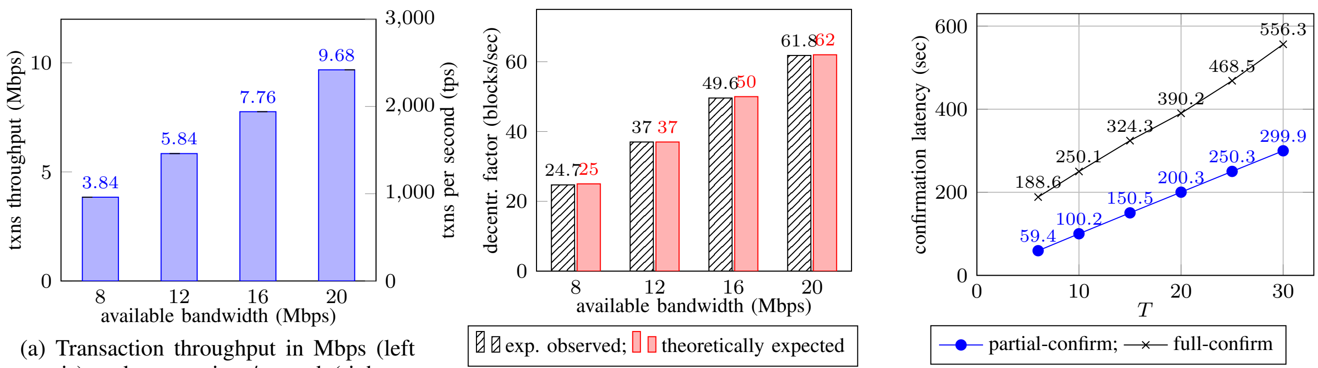
At the network level, a key difference between OHIE and other high-throughput protocols (e.g., Algorand [20]) is that with its large number of parallel chains, OHIE propagates a large number of small blocks in parallel. In contrast, protocols such as Algorand usually propagate a single large block (e.g., of 1 MB).

Now for OHIE, the critical empirical question is whether propagating many parallel blocks will have significant negative impact on BPD, as compared to propagating a single such block. We use micro experiments to answer this critical question. Figure 4(left) plots the BPD for 20 KB blocks (results for block sizes ranging from 10-64KB are similar and not shown), as a function of number of parallel block propagations. For example, "40 parallel blocks/sec" means that we inject 40 new blocks (of size 20 KB each) into the network per second. The figure shows that under 20Mbps raw bandwidth, even 60 parallel blocks per second will not cause any substantial increase in BPD.

Figure 4(right) presents the same results from a different perspective — it plots how BPD changes as the fraction of raw bandwidth used by the parallel block propagation increases. It shows that, consistently under all our bandwidth configurations, parallel block propagation can effectively utilize a rather significant fraction (about 50%) of the raw bandwidth, without significant negative impact on BPD. This simple yet important finding lays the empirical foundation for parallel chain designs. In particular, we can hope OHIE to eventually achieve a throughput approaching a significant fraction of the raw network bandwidth, by using a sufficient number of parallel chains.

C. End-to-end Performance of OHIE

We finally use macro experiments to evaluate the end-to-end performance of OHIE, in terms of its throughput, decentralization factor, and confirmation latency. By default, all our results will be from running 12,000 nodes on 1000 EC2 instances, with different per-node bandwidth configurations (8-20Mbps). We have also experimented with 50,000 nodes on



(a) Transaction throughput in Mbps (left y-axis) and transactions/second (right y-axis).

(b) Decentralization factor.

(c) Confirmation latency.

Fig. 5: OHIE's throughput, decentralization factor, and confirmation latency.

1000 EC2 instance, with 20Mbps per-node bandwidth. Those results are within 1% of the results under the corresponding experiment with $12,000$ node, and we do not report those separately.

We always use 20 KB blocks, with block interval being 10 seconds on each chain, as determined in Section VI-A. We choose k according to the available bandwidth, such that $k \times$ block_size/block_interval $\approx 0.5 \times$ available_bandwidth. Specifically, for 8Mbps, 12Mbps, 16Mbps, and 20Mbps per-node available bandwidth, we use $k = 250$, 370, 500, and 620, respectively. Since these k values are not powers of 2, we do not use the last $\log_2 k$ bits of the block hash to decide which chain a block belongs to. Instead, let x be the last 48 bits of the block hash, and we assign the block to chain i where $i = x \mod k$. Assuming the hash function is a random oracle, doing so will assign each block to a uniformly random chain, except some negligible probability.[18] Finally, to be able to run multiple nodes on each EC2 instance, we do not have the nodes solve PoW puzzles. Instead, each node produces ("mines") a new block after some exponentially distributed time that corresponds to the mining difficulty.

Throughput. Figure 5a shows that the throughput of OHIE indeed scales up roughly linearly with the available bandwidth. In fact, the throughput of OHIE always reaches a rather significant fraction (about 50%) of the raw available network bandwidth of the system. Under 20Mbps available bandwidth, the throughput of OHIE is about 9.68Mbps, or about $2,420$ transactions per second (assuming 500-byte average transaction size as in Bitcoin). As a quick comparison, OHIE achieves about 550% of the throughput of AlgoRand [20] under similar available bandwidth. Compared to Conflux under 20Mbps available bandwidth [32], the throughput of OHIE is about 150% of the throughput of Conflux. This suggests that while explicitly focusing on simplicity, OHIE still retains the high throughput property of modern blockchain designs.

Decentralization factor. Another advantage of OHIE is its *decentralization factor*, in terms of the number of distinct confirmed blocks per second. Figure 5b shows that the decentralization factor of OHIE increases linearly with the available bandwidth. This is expected since the number of parallel blocks (or chains) increases with the available bandwidth. In particular, OHIE achieves a decentralization factor of about 61.8 under 20Mbps available bandwidth. This is at least about 20x higher than those reported in experiments on previous permissionless protocols, among which Omniledger [28] reported the best decentralization factor of about 3.1 (i.e., 25 blocks in 8.1 seconds).

Confirmation latency. Let T be the number of blocks that we remove from the end of the chain in order to obtain partially-confirmed blocks in each individual chain in OHIE. For example, Bitcoin and Ethereum use $T = 6$ and $T = 10$ to 15, respectively. For comparable security, our analysis in Section V suggests using a T that is $\Theta(\log k)$ larger. Given that our k is no larger than 2^{14}, we use $T = 20$ to 30 in OHIE. Note that T has impact only on confirmation latency, and has no impact on the throughput or decentralization factor of OHIE.

Figure 5c plots the average time for a block to become partially-confirmed and fully-confirmed on all nodes, under 20Mbps bandwidth configuration. It shows that a block takes about 1-5 minutes to become partially-confirmed, and then about another 2-4 minutes to become fully-confirmed. As a reference point, the confirmation latencies in Bitcoin and Ethereum are presently 60 and 3 minutes, respectively.

For a conservative $T = 30$, we have done further experiments confirming (not plotted in Figure 5c) that the partial and full confirmation latencies remain stable under various bandwidth configurations from 8Mbps to 20Mbps. Putting it another way, the latency does not deteriorate as the throughput of OHIE increases.

VII. RELATED WORK

OHIE uses PoW under the same permissionless model as Nakamoto consensus. There have been works in alternative models, such as using Proof-of-Stake [7], [8], [20], [25] or assuming a permissioned setting [11], [16], [38]. PoW-based

[18]This is not exactly uniformly random since the range of x is not an exact multiple of k. But the impact is negligible.

permissionless blockchain protocols have largely followed two paradigms: The first based on extensions to Nakamoto consensus, and the second based on utilizing classical byzantine agreement (BA). We discuss these two categories one by one.

Nakamoto consensus. Existing deployments of Nakamoto consensus (e.g., Bitcoin) and its variant the GHOST protocol [44] (e.g., Ethereum) achieve only about 5KByte/s throughput. Bitcoin-NG is a variant of Nakamoto consensus, where many micro-blocks are proposed by the same proposer, after the proposer is chosen by a key block [13]. This helps to significantly improve throughput without comprising security, but at the same time, still has limited decentralization: A single block proposer is responsible for generating all (micro) blocks for an extended period of time (i.e., 10 minutes), inviting censorship attacks and DoS attacks. In comparison, OHIE achieves superior decentralization — in our experiments, in each second, up to about 60 different proposers propose blocks concurrently.

Scaling Nakamoto consensus. Phantom [45] and Conflux [32] have attempted to scale Nakamoto consensus, by having blocks reference more than one previous blocks. Section I has already discussed these two works.

Chainweb [35], [42] is another attempt to scale Nakamoto consensus, by maintaining k parallel chains. Unlike OHIE, Chainweb does not have any mechanism to prevent the adversary from focusing entirely on one of the k chains. Chainweb's original analysis [35], [42] only considers a few specific attack strategies [40], instead of all possible adversaries. Fitzi et al. [15] have shown that an adversary focusing on a specific chain can cause the confirmation latency in Chainweb to increase *quadratically* in k. In contrast, OHIE does not suffer from such a problem. Furthermore, different from OHIE where the `rank` values balance all the chains, Chainweb requires synchronous growth of all chains, needing to periodically stall fast-growing chains. Kiffer et al. [26] have provided a further analysis of Chainweb, which does not support the high-throughput claim by Chainweb. In their analysis, a natural form of Chainweb is "bounded by the same throughput as Nakamoto protocol for the same consistency guarantee" [26].

Concurrent with and independent of this work, there are two online non-refereed technical reports [5], [15] proposing a similar approach to composing multiple parallel chains. While their high-level designs share similarities with ours, in their designs, one of the k parallel chains is designated as a special chain, and blocks on the other chains are related to blocks on that special chain [5], [15]. In OHIE, all k chains are equal and symmetric. More importantly, these two concurrent and independent works [5], [15] do not have any implementation details or experimental evaluation, while we have a prototype implementation as well as large-scale evaluation on Amazon EC2. In particular, our large-scale experiments confirm that propagating many parallel blocks does *not* negatively impact block propagation delay, which lays the empirical foundation for parallel chain designs.

Finally, Spectre [43] confirms blocks without guaranteeing a total order, while the inclusive protocol [31] includes as many non-conflicting transactions as possible. These works provide weaker consistency notions than OHIE, which guarantees a total order. These weaker notions may suffice for cryptocurrency payments, but for smart contracts, total ordering is key in resolving state conflicts [29], [30] and in building towards higher abstractions of consistency [3], [12], [28].

Blockchain protocols relying on byzantine agreement. Some PoW-based permissionless blockchain protocols [2], [12], [27], [28], [33], [41], [47] build upon classical byzantine agreement (BA) protocols. BA protocols require a committee of pre-agreed identities among which the BA protocol can run. Such a committee can either be established using Nakamoto consensus itself or using previous rounds of BA.

Different from OHIE and other Nakamoto-style protocols, which can tolerate any constant $f < \frac{1}{2}$, BA-based blockchain protocols [2], [20], [27], [33], [41], [47] all require $f < \frac{1}{3}$. A key bottleneck in BA-based designs, as acknowledged in prior works, is that of establishing (or replenishing) committees with 200-2000 identities each. This large committee size is necessary to ensure the requisite resilience f. The latency of replenishing committees can be between in tens of seconds [2], [20], minutes [33], [47], or hours [28]. After committees are established and when there are no attacks, BA protocols tend to have smaller confirmation latencies [20], [27], [47] than Nakamoto-style protocols. Finally, we point out that classical BA protocols, such as PBFT [9], can be considerably more complex to implement and verify, as compared to Nakamoto consensus.

Sharding designs. A subset of the BA-based blockchain protocols employ sharding [28], [33], [47], where many parallel shards process blocks in parallel. This helps to improve decentralization and throughput, at the cost of additional complexity. But due to the overheads associated with each shard, these protocols [28], [33], [47] typically use a small number of shards (no more than 25). The best decentralization was achieved in [28], with about 25 blocks proposed every 8 seconds. In comparison, OHIE does not involve pre-assigning nodes to its k chains, and can easily use as large a k as needed. The decentralization factor of OHIE in our experiments (i.e., about 600 blocks every 10 seconds) is about 20x higher than the sharding designs.

VIII. CONCLUSION

We present OHIE, a permissionless proof-of-work blockchain protocol that composes parallel instances of Nakamoto consensus securely. OHIE has a simple implementation and a modular safety and liveness proof. It achieves linear scaling with available bandwidth and at least about 20x better decentralization over prior works.

ACKNOWLEDGMENTS

We thank Hung Dang, Seth Gilbert, Aquinas Hobor, Ilya Sergey, Shruti Tople, and Muoi Tran for their helpful feedbacks on this work. We thank Sourav Das for help on formatting the figures in this paper. This work is partly supported by the sponsors of the Crystal Center at National University of Singapore. All opinions and findings presented in this work are those of the authors only.

References

[1] OHIE - Blockchain scaling. https://github.com/ivicanikolicsg/OHIE, 2019.

[2] Abraham, I., Malkhi, D., Nayak, K., Ren, L., and Spiegelman, A. Solida: A blockchain protocol based on reconfigurable byzantine consensus. In *International Conference on Principles of Distributed Systems (OPODIS)* (2017).

[3] Al-Bassam, M., Sonnino, A., Bano, S., Hrycyszyn, D., and Danezis, G. Chainspace: A sharded smart contracts platform. In *Network and Distributed System Security Symposium (NDSS)* (2018).

[4] Apostolaki, M., Zohar, A., and Vanbever, L. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *IEEE Symposium on Security and Privacy* (2017).

[5] Bagaria, V., Kannan, S., Tse, D., Fanti, G., and Viswanath, P. Deconstructing the blockchain to approach physical limits. *arXiv preprint arXiv:1810.08092v2* (2018).

[6] Bailis, P., and Ghodsi, A. Eventual consistency today: limitations, extensions, and beyond. *Commun. ACM* (2013).

[7] Bentov, I., Pass, R., and Shi, E. Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive* (2016).

[8] Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J. A., and Felten, E. W. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *IEEE Symposium on Security and Privacy* (2015).

[9] Castro, M., and Liskov, B. Practical byzantine fault tolerance. In *Symposium on Operating Systems Design and Implementation (OSDI)* (1999).

[10] Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A. E., Miller, A., Saxena, P., Shi, E., Sirer, E. G., Song, D., and Wattenhofer, R. On scaling decentralized blockchains - (A position paper). In *Financial Cryptography and Data Security - FC International Workshops (BITCOIN)* (2016).

[11] Danezis, G., and Meiklejohn, S. Centrally banked cryptocurrencies. In *Network and Distributed System Security Symposium (NDSS)* (2016).

[12] Decker, C., Seidel, J., and Wattenhofer, R. Bitcoin meets strong consistency. In *International Conference on Distributed Computing and Networking (ICDCN)* (2016).

[13] Eyal, I., Gencer, A. E., Sirer, E. G., and Van Renesse, R. Bitcoin-NG: A Scalable Blockchain Protocol. In *NSDI* (2016).

[14] Eyal, I., and Sirer, E. G. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM 61*, 7 (2018), 95–102.

[15] Fitzi, M., Gazi, P., Kiayias, A., and Russell, A. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. Cryptology ePrint Archive, report 2018/1119, version 20181130:165620, Nov. 2018. https://eprint.iacr.org/2018/1119.

[16] Garay, J., and Kiayias, A. Sok: A consensus taxonomy in the blockchain era, 2018. https://eprint.iacr.org/2018/754.pdf.

[17] Garay, J., Kiayias, A., and Leonardos, N. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT* (2015).

[18] Garay, J. A., Kiayias, A., and Leonardos, N. The bitcoin backbone protocol with chains of variable difficulty. In *CRYPTO* (2017).

[19] Gencer, A. E., Basu, S., Eyal, I., van Renesse, R., and Sirer, E. G. Decentralization in bitcoin and ethereum networks. In *Financial Cryptography and Data Security (FC)* (2018).

[20] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Symposium on Operating Systems Principles* (2017).

[21] Gilbert, S., and Lynch, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News 33*, 2 (2002), 51–59.

[22] Gilbert, S., and Lynch, N. A. Perspectives on the CAP theorem. *IEEE Computer 45*, 2 (2012), 30–36.

[23] Heilman, E., Kendler, A., Zohar, A., and Goldberg, S. Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX Security Symposium* (2015).

[24] Kiayias, A., and Panagiotakos, G. Speed-security tradeoffs in blockchain protocols. *IACR Cryptology ePrint Archive 2015* (2015), 1019.

[25] Kiayias, A., Russell, A., David, B., and Oliynykov, R. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO* (2017).

[26] Kiffer, L., Rajaraman, R., and abhi shelat. A Better Method to Analyze Blockchain Consistency. In *CCS* (2018).

[27] Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., and Ford, B. Enhancing bitcoin security and performance with strong consistency via collective signing. In *USENIX Security Symposium* (2016).

[28] Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., and Ford, B. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *IEEE Symposium on Security and Privacy* (2018).

[29] Kung, H. T., and Papadimitriou, C. H. An optimality theory of concurrency control for databases. In *SIGMOD* (1979).

[30] Lamport, L. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Computers 28*, 9 (1979), 690–691.

[31] Lewenberg, Y., Sompolinsky, Y., and Zohar, A. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security* (2015).

[32] Li, C., Li, P., Zhou, D., Xu, W., Long, F., and Yao, A. Scaling nakamoto consensus to thousands of transactions per second, 2018. Arxiv preprint, arXiv:1805.03870v4.

[33] Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., and Saxena, P. A secure sharding protocol for open blockchains. In *CCS* (2016).

[34] Luu, L., Velner, Y., Teutsch, J., and Saxena, P. Smartpool: Practical decentralized pooled mining. In *USENIX Security Symposium* (2017).

[35] Martino, W., Quaintance, M., and Popejoy, S. Chainweb: A Proof-of-Work Parallel-Chain Architecture for Massive Throughput (DRAFT v15), 2018. https://kadena.io/download/178/.

[36] McCorry, P., Hicks, A., and Meiklejohn, S. Smart contracts for bribing miners. *IACR Cryptology ePrint Archive* (2018).

[37] Merkle, R. C. A digital signature based on a conventional encryption function. In *CRYPTO* (1987).

[38] Miller, A., Xia, Y., Croman, K., Shi, E., and Song, D. The honey badger of BFT protocols. In *CCS* (2016).

[39] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system, 2008. https://bitcoin.org/bitcoin.pdf.

[40] Pass, R., Seeman, L., and Shelat, A. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT* (2017).

[41] Pass, R., and Shi, E. Hybrid consensus: Efficient consensus in the permissionless model. In *International Symposium on Distributed Computing* (2017).

[42] Quaintance, M., and Martino, W. Chainweb Protocol Security Calculations (WORK IN PROGRESS - DRAFT v7), 2018. https://kadena.io/download/187/.

[43] Sompolinsky, Y., Lewenberg, Y., and Zohar, A. Spectre: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive* (2016).

[44] Sompolinsky, Y., and Zohar, A. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security* (2015).

[45] Sompolinsky, Y., and Zohar, A. Phantom: A scalable blockdag protocol, 2018. https://eprint.iacr.org/2018/104/20180330:121321.

[46] Yu, H., Nikolic, I., Hou, R., and Saxena, P. OHIE: Blockchain Scaling Made Simple, 2019. Arxiv preprint, arXiv:1811.12628.

[47] Zamani, M., Movahedi, M., and Raykova, M. RapidChain: Scaling Blockchain via Full Sharding. In *CCS* (2018).

Appendix

A. Proof for Lemma 4

Lemma 4 directly follows from Lemma 6 and Lemma 7, which we will state and prove next. To prove Lemma 6 and Lemma 7, we will need Lemma 5. Lemma 5 shows that except some exponentially small probability, given a block B and regardless of whether B is an honest block or malicious block, the value of $\widehat{B}$ (and in particular, the value of $\widehat{B}.\texttt{rank}$

/ $\widehat{B}$.next_rank) must be the same on all honest nodes. This avoids the need of reasoning about potentially different $\widehat{B}$.rank ($\widehat{B}$.next_rank) values on different honest nodes. In the following, we will state and prove Lemma 5 through 7, one by one.

Lemma 5. *Consider the execution of (k, p, λ, T)-OHIE against any given adversary $\mathcal{A}$. With probability at least $1 - exp(-\Omega(\lambda))$, there will never be two honest nodes u_1 and u_2 adding $(B_1, \widehat{B_1})$ and $(B_2, \widehat{B_2})$ to their local set of blocks, respectively, such that $B_1 = B_2$ and $\widehat{B_1} \neq \widehat{B_2}$.*

Proof. Define bad1 to be the event that the execution (of (k, p, λ, T)-OHIE against $\mathcal{A}$) contains some invocations $hash(x_1)$ and $hash(x_2)$ such that $x_1 \neq x_2$ and $hash(x_1) = hash(x_2)$. Given that we model the hash function as a random oracle, and given that the length of the execution is poly(λ), we have $\Pr[\texttt{bad1}] = exp(-\Omega(\lambda))$.

Next define bad2 to be the event that the execution (of (k, p, λ, T)-OHIE against $\mathcal{A}$) contains some invocation of $hash()$ that belongs to one of the following two categories:

- For some B, some honest node invokes $hash(B)$ at Line 42 of Figure 3 and the verification at that line succeeds (i.e., $hash(B)$ indeed equals $\widehat{B}$.hash), despite that $hash(B)$ has never been previous invoked (by either some honest node or the adversary) in the execution.
- Some honest node invokes $hash()$ at Line 43 of Figure 3 and the Merkle verification at that line succeeds, despite the following: Let $hash(x_1)$, $hash(x_2)$, ..., $hash(x_{\log_2 k})$ be the $\log_2 k$ hash invocations done by the honest node during the successful Merkle verification. There exists some x_i ($1 \leq i \leq \log_2 k$) such that $hash(x_i)$ has never been previously invoked (by either some honest node or the adversary) in the execution.

Again given that we model the hash function as a random oracle, and given that the length of the execution is poly(λ), one can easily verify that $\Pr[\texttt{bad2}] = exp(-\Omega(\lambda))$.

A simple union bound then shows that with probability at least $1 - exp(-\Omega(\lambda))$, neither bad1 nor bad2 happens. It now suffices to prove that conditioned upon neither of the bad events happening, if $B_1 = B_2$, then we must have $\widehat{B_1} = \widehat{B_2}$. An attachment consists of five fields, namely, hash, leaf, leaf_proof, rank, and next_rank. We will reason about these one by one.

Since $B_1 = B_2$, and since $(B_1, \widehat{B_1})$ and $(B_2, \widehat{B_2})$ have been accepted by u_1 and u_2 respectively, we trivially have $\widehat{B_1}$.hash $= \widehat{B_2}$.hash. We next prove $\widehat{B_1}$.leaf $= \widehat{B_2}$.leaf and $\widehat{B_1}$.leaf_proof $= \widehat{B_2}$.leaf_proof. Since u_1 has verified the Merkle proof before accepting $(B_1, \widehat{B_1})$, u_1 must have invoked $hash(x)$ for some x of length 2λ, with the return value of such invocation being B_1.root. Since bad1 and bad2 do not happen, such an x must be unique. Let $x_0|x_1 \leftarrow x$, where x_0 and x_1 are both of length λ. Following this process for $\log_2 k$ steps (we are effectively tracing down the Merkle tree), we will find a unique $x_\perp$ value, which corresponds to leave i of the Merkle tree, with i being the last $\log_2 k$ bits of $\widehat{B_1}$.hash. Since bad1 and bad2 do not happen, in order for $(B_1, \widehat{B_1})$ to pass the verification by u_1, we must have $\widehat{B_1}$.leaf $= x_\perp$. By the same argument and since B_1.root $= B_2$.root, we must also have $\widehat{B_2}$.leaf $= x_\perp$. Hence we conclude $\widehat{B_1}$.leaf $= \widehat{B_2}$.leaf. A similar argument shows that $\widehat{B_1}$.leaf_proof $= \widehat{B_2}$.leaf_proof as well.

Next we move on to the rank and next_rank fields. Let A_1 be any block on u_1 such that $\widehat{A_1}$.hash $= x_\perp$, where $x_\perp$ is obtained as above. Similarly define A_2. Since bad1 and bad2 do not happen, we must have $A_1 = A_2$. Next, let C_1 be any block on u_1 such that $\widehat{C_1}$.hash $= B_1$.trailing, and let C_2 be any block on u_2 such that $\widehat{C_2}$.hash $= B_2$.trailing $= B_1$.trailing. Similarly, we must also have $C_1 = C_2$.

Consider the DAG $\mathcal{G}_1$ consisting of all the blocks on u_1 as vertices, where for each block B_1, there is an edge to B_1 from the corresponding A_1, and another edge to B_1 from the corresponding C_1. We do a topological sort of all the vertices in $\mathcal{G}_1$, and assume that B_1 is the jth block in the topological sort. (Note that j is based on B_1 and $\mathcal{G}_1$, and has nothing to do with B_2.) We will do an induction on j to show that for all $B_1 = B_2$, we have $\widehat{B_1}$.rank $= \widehat{B_2}$.rank and $\widehat{B_1}$.next_rank $= \widehat{B_2}$.next_rank.

The induction basis for j from 1 to k trivially holds (these are the k genesis blocks). Now assume that the previous claim holds for all $j < j_1$, and we prove the claim for $j = j_1$. Since B_1 is the j_1-th block in the topological sort, the position of A_1 and C_1 in the topological sort must be before j_1. Furthermore, we have shown earlier that $A_1 = A_2$ and $C_1 = C_2$. We can thus invoke the inductive hypothesis on A_1 and C_1, which shows that $\widehat{A_1}$.next_rank $= \widehat{A_2}$.next_rank and $\widehat{C_1}$.next_rank $= \widehat{C_2}$.next_rank. By Line 49 in Figure 3, $\widehat{B_1}$.rank is set to be the same as $\widehat{A_1}$.next_rank, while $\widehat{B_2}$.rank is set to be the same as $\widehat{A_2}$.next_rank. Hence we have $\widehat{B_1}$.rank $= \widehat{B_2}$.rank. A similar argument shows $\widehat{B_1}$.next_rank $= \widehat{B_2}$.next_rank. ☐

Lemma 6. *If the three properties in Theorem 2 hold for each of the k chains in (k, p, λ, T)-OHIE, then with probability at least $1 - exp(-\Omega(\lambda))$, (k, p, λ, T)-OHIE satisfies the* **consistency** *property in Theorem 1.*

Proof. Lemma 5 shows that with probability at least $1 - exp(-\Omega(\lambda))$, there will never be two honest nodes adding $(B_1, \widehat{B_1})$ and $(B_2, \widehat{B_2})$ to their respective local set of blocks, such that $B_1 = B_2$ and $\widehat{B_1} \neq \widehat{B_2}$. This means that for each block B accepted by an honest node, its rank and next_rank on this honest node will be the same as the corresponding values on all other honest nodes. Hence we can directly refer to the rank and next_rank of a block B, and no longer need to consider the values of $\widehat{B}$.rank and $\widehat{B}$.next_rank on individual nodes. All our following discussion will be conditioned on this.

The following restates the **consistency** property in Theorem 1, which we need to prove:

- **(consistency)** Consider the SCB S_1 on any node u_1 at any time t_1, and the SCB S_2 on any node u_2 at any time

t_2.[19] Then either S_1 is a prefix of S_2 or S_2 is a prefix of S_1. Furthermore, if ($u_1 = u_2$ and $t_1 < t_2$) or ($u_1 \neq u_2$ and $t_1 + \Delta < t_2$), then S_1 is a prefix of S_2.

Let the view of node u_1 at time t_1 be Ψ_1, and the view of node u_2 at time t_2 be Ψ_2. Let x_1 and x_2 be the `confirm_bar` in Ψ_1 and Ψ_2, respectively. Without loss of generality, assume $x_1 \leq x_2$. Let S_1 and S_2 be the SCB in Ψ_1 and Ψ_2, respectively. The next will prove that S_1 is a prefix of S_2. In the proof, we will sometimes use set operations over S_1 and S_2. For example, $S_1 \cap S_2$ refers to the set of common blocks in S_1 and S_2.

Let $F_1(i)$ be the sequence of partially-confirmed blocks on chain i in Ψ_1. Let $G_1(i)$ be the prefix of $F_1(i)$ such that $G_1(i)$ contains all blocks in $F_1(i)$ whose `rank` is smaller than x_1. Similarly define $F_2(i)$ and $G_2(i)$, where $G_2(i)$ contains those blocks in $F_2(i)$ whose `rank` is smaller than x_2. We first prove the following two claims:

- For all i where $0 \leq i \leq k-1$, $G_1(i)$ is a prefix of $G_2(i)$. To prove this claim, note that by the **consistency** property in Theorem 2, either $F_1(i)$ is a prefix of $F_2(i)$ or $F_2(i)$ is a prefix of $F_1(i)$. If $F_1(i)$ is a prefix of $F_2(i)$, then together with the fact that $x_1 \leq x_2$, it is obvious that $G_1(i)$ is a prefix of $G_2(i)$. If $F_2(i)$ is a prefix of $F_1(i)$, let x_3 be the `rank` of the last block in $F_2(i)$. This also means that the `next_rank` of that block is at least $x_3 + 1$. By our design of `confirm_bar`, we know that $x_2 \leq x_3 + 1$. In turn, we have $x_1 \leq x_2 \leq x_3 + 1$. Hence $G_1(i)$ must also be a prefix of $G_2(i)$.
- For all i where $0 \leq i \leq k-1$ and all block $B \in G_2(i) \setminus G_1(i)$, B's `rank` must be no smaller than x_1. We prove this claim via a contradiction and assume that B's `rank` is smaller than x_1. Together with the fact that B is in $G_2(i) \setminus G_1(i)$, we know that B is in $F_2(i)$ but not in $F_1(i)$. Hence $F_2(i)$ cannot be a prefix of $F_1(i)$. Then by the **consistency** property in Theorem 2, $F_1(i)$ must be a prefix of $F_2(i)$. Let x_4 be the `rank` of the last block D in $F_1(i)$. Since $F_1(i)$ is a prefix of $F_2(i)$, both D and B must be in $F_2(i)$, and D must be before B in $F_2(i)$. Since the blocks in $F_2(i)$ must have increasing `rank` values, we know that D's `rank` must be smaller than B's. Hence we have $x_4 = D$'s `rank` $< B$'s `rank` $\leq x_1 - 1$, or more concisely, $x_4 < x_1 - 1$. On the other hand, since D's `rank` is x_4, its `next_rank` must be at least x_4+1. By our design of `confirm_bar` in Ψ_1, we know that $x_1 \leq x_4+1$ and hence $x_4 \geq x_1 - 1$. This yields a contradiction.

Now we can use the above two claims to prove that S_1 is a prefix of S_2. S_1 consists of all the blocks in $G_1(0)$ through $G_1(k-1)$, while S_2 consists of all the blocks in $G_2(0)$ through $G_2(k-1)$. Since $G_1(i)$ is a prefix of $G_2(i)$ for all i, we know that $S_1 \subseteq S_2$. For all blocks in S_1 (which is the same as $S_2 \cap S_1$), the sequence S_1 orders them in exactly the same way as the sequence S_2. For all block $B \in S_2 \setminus S_1$, by the second claim above, we know that B's `rank` must be no smaller than x_1. Hence in the sequence S_2, all blocks in $S_2 \setminus S_1$ must be ordered after all the blocks in $S_2 \cap S_1$ (whose `rank` must be smaller than x_1). This completes our proof that S_1 is a prefix of S_2.

Finally, if $u_1 = u_2$ and $t_1 < t_2$, then since `confirm_bar` on a node never decreases over time, we must have $x_1 \leq x_2$. Similarly, if $u_1 \neq u_2$ and $t_1 + \Delta < t_2$, then by time t_2, node u_2 must have seen all blocks seen by u_1 at time t_1. By the **consistency** property in Theorem 2, all partially-confirmed blocks on u_1 at time t_1 must also be partially-confirmed on u_2 at time t_2. Thus we must also have $x_1 \leq x_2$. Putting everything together, if ($u_1 = u_2$ and $t_1 < t_2$) or ($u_1 \neq u_2$ and $t_1 + \Delta < t_2$), then S_1 must be a prefix of S_2. ☐

Lemma 7. *If the three properties in Theorem 2 hold for each of the k chains in (k, p, λ, T)-OHIE, then with probability at least $1 - exp(-\Omega(\lambda))$, (k, p, λ, T)-OHIE satisfies the* **quality-growth** *property in Theorem 1.*

Proof. Same as the reasoning in the beginning of the proof for Lemma 6, we first invoke Lemma 5 to show that with probability at least $1 - exp(-\Omega(\lambda))$, there will never be two honest nodes adding $(B_1, \widehat{B_1})$ and $(B_2, \widehat{B_2})$ to their respective local set of blocks, such that $B_1 = B_2$ and $\widehat{B_1} \neq \widehat{B_2}$. All our following discussion will be conditioned on this, and we will be able to directly refer to the `rank` and `next_rank` of a block B.

The following restates the **quality-growth** property in Theorem 1, which we need to prove:

- **(quality-growth)** For all integer $\gamma \geq 1$, the following property holds after the very first $\frac{2T}{pn}$ ticks of the execution: On any honest node, in every $(\gamma+2) \cdot \frac{2T}{pn} + 2\Delta$ ticks, at least $\gamma \cdot k \cdot \frac{1-2f}{1-f} T$ honest blocks are newly added to SCB.

Consider any given honest node u, and any given time t_0 (in terms of ticks from the beginning of the execution), where t_0 is after the very first $\frac{2T}{pn}$ ticks of the execution. By the **growth** property in Theorem 2, at time t_0, the length of every chain in (k, p, λ, T)-OHIE must be at least T. By the **growth** property and the **quality** property in Theorem 2, we know that from time t_0 to time $t_1 = t_0 + \gamma \cdot \frac{2T}{pn}$ on node u, every chain in (k, p, λ, T)-OHIE has at least $\gamma \cdot \frac{1-2f}{1-f} T$ honest blocks becoming newly partially-confirmed. Let α_i denote the set of such newly partially-confirmed honest blocks on chain i (for $0 \leq i \leq k-1$). We have $|\alpha_i| \geq \gamma \cdot \frac{1-2f}{1-f} T$ for all i, and $| \cup_{i=0}^{k-1} \alpha_i| \geq \gamma \cdot k \cdot \frac{1-2f}{1-f} T$. To prove the lemma, it suffices to show that by time $t_4 = t_0 + (\gamma+2) \cdot \frac{2T}{pn} + 2\Delta$ on node u, all blocks in α_i will have become fully-confirmed for all $0 \leq i \leq k-1$. Without loss of generality, we only need to prove for α_0.

Let x be the largest `next_rank` among all the blocks in α_0. By definition of such x, the `rank` value of every block in α_0 must be smaller than x. Let y be the length of chain 0 on node u at time t_1. By time $t_2 = t_1 + \Delta$, all honest nodes will have received all blocks in α_0. Furthermore, the length of chain 0 on all honest nodes at time t_2 must be at least y. Together with the **consistency** property in Theorem 2, we know that all the

[19]Here u_1 (t_1) may or may not equal u_2 (t_2).

blocks in α_0 must be partially-confirmed on all honest nodes by time t_2. Thus starting from t_2, whenever an honest node (including node u) mines a block, by the design of OHIE, the `next_rank` of the new honest block will be at least x. We will invoke this important property later.

Next let us come back to the honest node u, and consider any one of the chains. From time $t_3 = t_2 + \Delta$ to time $t_4 = t_3 + \frac{4T}{pn}$ on node u, by the **growth** property in Theorem 2, the length of this chain must have increased by at least $2T$ blocks. The first T blocks among all these blocks must have been partially-confirmed on node u at time t_4. By the **quality** property in Theorem 2, these first T blocks must contain at least $\frac{1-2f}{1-f}T$ honest blocks. For $T \geq \frac{1-f}{1-2f}$, these first T blocks must contain at least one honest block B that is partially-confirmed. Since B is first seen by u no earlier than t_3, we know that this honest block B must have been generated (either by u or by some other honest node) no earlier than t_2. By our earlier argument, the `next_rank` of B must be at least x.

Finally, note that we actually have one such B on *every* chain on node u at time t_4. This means that on node u at time t_4, the `confirm_bar` is at least x. We earlier showed that for all blocks in α_0, their `rank` values must all be smaller than x. Hence such a `confirm_bar` enables all blocks in α_0 to become fully-confirmed on node u at time t_4. Observe that $t_4 = t_0 + (\gamma + 2) \cdot \frac{2T}{pn} + 2\Delta$, and we are done. $\square$