Scaling Nakamoto Consensus to Thousands of Transactions per Second

Chenxing Li, Peilun Li*, Wei Xu, Fan Long‡, and Andrew Chi-chih Yao*
Institute for Interdisciplinary Information Sciences, Tsinghua University
Computer Science Department, University of Toronto‡

Abstract

This paper presents Conflux, a fast, scalable and decentralized blockchain system that optimistically process concurrent blocks without discarding any as forks. The Conflux consensus protocol represents relationships between blocks as a direct acyclic graph and achieves consensus on a total order of the blocks. Conflux then, from the block order, deterministically derives a transaction total order as the blockchain ledger. We evaluated Conflux on Amazon EC2 clusters with up to 20k full nodes. Conflux achieves a transaction throughput of 5.76GB/h while confirming transactions in 4.5-7.4 minutes. The throughput is equivalent to 6400 transactions per second for typical Bitcoin transactions. Our results also indicate that when running Conflux, the consensus protocol is no longer the throughput bottleneck. The bottleneck is instead at the processing capability of individual nodes.

1 Introduction

Following the success of the cryptocurrencies [4, 20], blockchain has recently evolved into a technology platform that powers secure, decentralized, and consistent transaction ledgers at Internet-scale. The ledger becomes a powerful abstraction and fuels innovations on real-world applications in financial systems, supply chains, and health cares [8, 9, 13], shifting the landscapes of the industries that worth hundreds of billions of dollars.

Blockchain platforms like Bitcoin [20] use *Nakamoto consensus* as their consensus protocols. This protocol usually organizes transactions into an ordered list of blocks (i.e., a blockchain), each of which contains multiple transactions and a link to its predecessor. To defend against Sybil attacks, everyone solves *proof-of-work* problems (e.g., finding partial hash collisions) to compete for the right of generating the next block. To prevent an attacker from reverting previous transactions, everyone agrees on the longest chain of blocks as the correct transaction history. Each newly generated block will be appended at the end of the longest chain to make the chain even longer and therefore harder to revert.

However, the performance bottleneck remains one of the most critical challenges of current blockchains. In the standard Nakamoto consensus, the performance is bottlenecked by the facts 1) that only one participant can win the competition and contribute to the blockchain, i.e., concurrent blocks are discarded as *forks*, and 2) that the slowness is essential to defend against adversaries [24, 26]. For example, Bitcoin generates one 1MB block every 10 minutes and can therefore only process 7 transactions per second. Furthermore, to obtain high confidence that a transaction is irreversible in Bitcoin, users typically have to wait for tens of blocks building on top of the enclosing block of the transaction. This causes hours of waiting before confirming a transaction. The insufficient throughput and long confirmation delay severely limit the adoptions of blockchain techniques, causing poor user experience, congested network, and skyrocketing transaction fees [3, 5].

Previous research focuses on reducing the participations of the consensus to improve the performance without compromising the security of the blockchains. For example, Bitcoin-NG [10] periodically elects a leader and allows the leader to dictate the transaction total order for a period of time. It improves the throughput but not the confirmation time of transactions. Several proposals elect a small set of participants as a committee to run Byzantine fault tolerance (BFT) to determine the transaction order [12, 16, 18, 19]. This solution may create undesirable hierarchies among protocol participants and compromise the decentralization of the blockchains.

*Equal contributions and ranked alphabetically.

1.1 Conflux

We present Conflux, a fast, scalable, and decentralized blockchain system that can process thousands of transactions per second while confirming each transaction in minutes. The core of Conflux is its novel consensus protocol that allows multiple participants to contribute to the Conflux blockchain concurrently while still being provably safe. The protocol enables significantly faster block generation and, in turn, enables higher throughputs and faster confirmations. In fact, the Conflux throughput is no longer limited by the consensus protocol, but by the processing capability of each individual node.

One observation that drives the design of Conflux is that standard Nakamoto consensus protocols preemptively define a restrictive total order of transactions when generating each block. This strategy introduces many false dependencies that lead to unnecessary forks. The key to our approach is to *defer the transaction total ordering and optimistically process concurrent transactions and blocks*. Given that the transactions rarely conflict in blockchains (particularly in cryptocurrencies), Conflux first optimistically assumes that transactions in concurrent blocks would not conflict with each other by default and therefore considers only explicit happens-before relationships specified by the block generators. This enables Conflux to operate with less constraints and efficiently achieve consensus on one block total order among many possibilities. Each participant in Conflux then deterministically derives the transaction total order from the agreed block total order, discarding conflicting and/or duplicated transactions. This lazy reconciliation provides the same external interface to the users.

When a participant node generates a new block in Conflux, the node identifies a parent (predecessor) block for the new block and creates a *parent edge* between these two blocks like Bitcoin. To incorporate contributions from concurrent blocks, the node also identifies all other blocks that have no incoming edge and creates *reference edges* from the new block to those blocks. Such reference edges represent that those blocks are generated before the new block. As a result, the edges between blocks now form a sophisticated direct acyclic graph (DAG) rather than a chain with potential forks [24, 25].

One challenge Conflux faces is how to agree on an irreversible block total order of the DAG that include concurrent blocks. Conflux addresses this challenge with its novel ordering algorithm. Given a *pivot chain* that starts from the genesis block and that contains only parent edges, the ordering algorithm deterministically partitions all blocks in the DAG into *epochs* using the pivot chain, topological sorts blocks in each epoch, and concatenates the sorting results of all epochs into the final total order following the happens-before order between epochs. The algorithm guarantees that if the pivot chain stabilizes (i.e., blocks on the chain are irreversible except the last few blocks), then the produced block total order will stabilize (i.e., blocks in the total order are irreversible except blocks in the last few epochs). With its ordering algorithm, Conflux reduces the problem of achieving consensus on a block total order of the DAG to the problem of achieving consensus on a pivot chain. Conflux therefore uses a modified chain-based Nakamoto consensus [26] to solve the chain consensus problem.

Assumptions and Guarantees: Conflux operates under similar assumptions as Bitcoin and many others. Honest nodes are reasonably synchronous respecting a network diameter d (i.e., messages sent by honest nodes will be delivered with a maximum delay of d). All honest nodes together control more block generation power than attackers. Under these assumptions, the Conflux consensus protocol guarantees that the agreed block total order is irreversible with very high probability.

Note that we focus on the consensus protocol design and implementation and the incentive mechanism is outside the scope of this paper. Therefore in this paper, we refer "honest nodes" as nodes that run bug-free programs that faithfully implement Conflux. We leave designing a compatible incentive mechanism to economically encourage honest behaviors as future work.

1.2 Experimental Results

We implemented a prototype of Conflux and evaluated Conflux by deploying up to 20k Conflux full nodes on 800 Amazon EC2 virtual machines. We also compared Conflux with two standard chain-based Nakamoto consensus approaches, Bitcoin [20] and GHOST [26].

Our experimental results show that under the 20Mbps bandwidth limit for each full node (i.e., the same experimental environment as Algorand [12]), Conflux can process one 4MB block per 5 seconds and therefore achieves the transaction throughput of 2.88GB/h; the Conflux throughput is 11.62x of the throughput of Bitcoin and GHOST, and is 3.84x of the throughput of Algorand [12]. Under the 40Mbps bandwidth limit, Conflux can process one 4MB block per 2.5 seconds and therefore achieves an even higher transaction throughput of 5.76GB/h. The Conflux throughput is equivalent to 6400 transactions per second for typical Bitcoin transactions. In fact, our results indicate that when running Conflux, the consensus protocol is no longer the bottleneck of the throughput,

but the processing capability of individual nodes (e.g., the hardware bandwidth limit).

Our experiments also show that by working with faster generation rates, Conflux allows blocks building on top of each other more quickly and therefore enables fast confirmations. Conflux confirms transactions in 4.5-7.4 minutes under the 4MB/2.5s and 40Mbps setting and 7.6-13.8 minutes under the 4MB/5s and 20Mbps setting.

1.3 Contribution

This paper makes the following contributions:

- **Consensus Protocol:** We present a novel, fast and scalable DAG-based Nakamoto consensus protocol, Conflux, to optimistically process concurrent blocks while lazily reconciling the transaction total order from an agreed block total order.
- **Ordering Algorithm:** Conflux contains a novel ordering algorithm, which given a stabilized chain as the input produces a stabilized block total order of the DAG. The ordering algorithm enables Conflux to reduce the DAG total order consensus problem to the chain-based Nakamoto consensus.
- **Conflux Implementation:** We present a prototype implementation of Conflux based on the Bitcoin core codebase [1]. To the best of our knowledge, Conflux is the first blockchain system that uses a DAG-based Nakamoto consensus protocol and that can process thousands of transactions per second.
- **Experimental Results:** We present a systematic evaluation of Conflux. Our results show that Conflux can achieve the transaction throughputs of 2.88GB/h and 5.76GB/h and confirm transactions with high confidence in 4.5-13.8 minutes.

The rest of this paper is organized as follows. Section 2 presents an overview of Conflux and an illustrative running example for Conflux. Section 3 presents the Conflux consensus protocol. Section 4 discusses our prototype implementation of Conflux. Section 5 presents our evaluation of Conflux. We discuss related work in Section 6 and conclude in Section 7.

2 Overview and Example

This section presents an overview of Conflux and a running example to illustrate the Conflux consensus protocol.

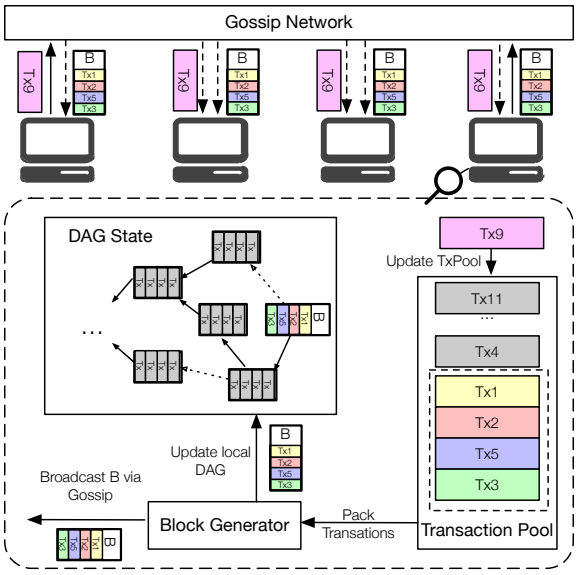

Figure 1: Architecture of Conflux.

2.1 Conflux Architecture

Figure 1 presents the architecture diagram of Conflux. Similar to Bitcoin, a transaction in Conflux is a payment message signed by the payer to transfer coins from the payer to the payee, where the payer and the payee are identified by their public keys. There can be special coinbase transactions to mint new coins. Each transaction also has a unique id generated by cryptographic digest functions to ensure its integrity. A block consists of a list of transactions (e.g. there are 4 transactions in block B of Figure 1) and reference links to previous blocks (e.g., in the DAG state in Figure 1, B links to two previous blocks). Each block also has a unique id generated by digest functions to ensure data integrity. At the very beginning, Conflux starts with a predefined *genesis block* to determine the initial state of the blockchain.

A key difference between Conflux and traditional blockchains is that the blocks and the edges form a DAG instead of a linear chain. As the blocks travel across the netowrk, each node might observe a slighlty different DAG due to network delays. The goal of Conflux is to maintain the local DAG of each individual node so that all nodes in Conflux can eventually agree on a total order of blocks (and transactions).

Note that having total orders in Conflux enables the supports of smart contracts, which execute Turing complete programs to update the state associated with accounts. For the purpose of illustration, in this paper we assume a balance model with simple payment transac-

tions. One transaction conflicts with previous transactions if after previous transactions, the payee does not have enough balance to execute the transaction.

Gossip Network: All participant nodes in Conflux are connected via a gossip network as shown in the top part of Figure 1. Whenever a node initiates a transaction, it broadcasts the transaction to all other nodes via the gossip network (e.g., Tx9 in Figure 1). Whenever a node generates a new block, it broadcasts the block to all other nodes via the network as well (e.g., the block B in Figure 1). Conflux currently uses a modified gossip network implementation in Bitcoin Core [1] (see Section 4).

Pending Transaction Pool: Each node maintains a pending transaction pool as shown in the bottom right of Figure 1. The pool contains all transactions that have been heard by the node but are not yet packed into any block. Whenever a node receives a new transaction from the gossip network, the node adds the transaction into its pool (e.g., adding Tx9 into the transaction pool in Figure 1). Whenever a node discovers a new block (either by generating the block or by receiving the block from other nodes), the node removes all transactions in the new block from its pending transaction pool. For example, in Figure 1, the node removes Tx1, Tx2, Tx3, and Tx5 from its pool after generating the block B. In Conflux concurrent blocks might pack duplicate or conflicting transactions due to network delays and malicious nodes. These transactions will be resolved by our consensus protocol.

Block Generator: Each node in Conflux runs a block generator to generate valid new blocks to pack pending transactions as shown in the bottom of Figure 1. Conflux operates with proof-of-work (PoW) mechanism, so the block generator attempts to find solutions for PoW problems to generate blocks. A valid block header must contain a PoW solution in its header. Once such a valid block header is generated, it selects pending transactions from the pool to fill the new block. Similar to Bitcoin, the PoW mechanism maintains a stable network block generation rate via dynamically adjusting the difficulty of the PoW problems. Note that besides PoW, Conflux can also work with any other mechanism that can maintain a stable block generation rate, such as proof-of-stake (PoS) [7, 15].

Local DAG State: Each node maintains a local state that contains all blocks which the node is aware of. Because in Conflux each block may contain links to reference several previous blocks not just one, the result state is a direct acyclic graph (DAG) as shown in center of Figure 1. Whenever the node discovers a new block (either by generating or by receiving it), the node updates its local DAG state accordingly.

2.2 Consensus Protocol

The Conflux consensus protocol operates on the local DAG state of each individual node. The goal of the protocol is to achieve the consensus on a total order of generated blocks among all nodes. From the total order of the blocks Conflux then derives a total order of transactions inside those blocks to process these transactions. For conflicting or duplicated transactions, Conflux only processes the first one and discards the remaining ones.

DAG and Edges: Figure 2 presents a running example of the local DAG state of a node in Conflux. We will use this example in the remaining of this section to illustrate the high level ideas of the Conflux consensus protocol. Each vertex in the DAG in Figure 2 corresponds to a block. In Figure 2, Genesis is the predefined genesis block. Only Genesis, A, B, and G are associated with transactions. There are two kinds of edges in the DAG, parent edges and reference edges:

- **Parent Edge:** Each block except Genesis has exactly one outgoing parent edge (solid line arrows in Figure 2). Intuitively, the parent edge corresponds to a voting relationship, i.e., the node that generates the child block votes for the transaction history represented by the parent block. For example, there are one parent edge from C to A and one from F to B.
- **Reference Edge:** Each block can have multiple outgoing reference edges (dashed lines arrows in Figure 2). A reference edge corresponds to generated-before relationships between blocks. For example, there is a reference edge from E to D. It indicates that D is generated before E.

Pivot Chain: Note that all parent edges in a DAG together form a *parental tree* in which the genesis block is the root. In the parental tree, Conflux selects a chain from the genesis block to one of the leaf blocks as the *pivot chain*. Unlike the Bitcoin protocol which selects the longest chain in the tree, Conflux selects the pivot chain based on the GHOST rule [26]. Specifically, the selection algorithm starts from the genesis block. At each step, it computes the subtree sizes of each child in the parental tree and advances to the child block with the largest subtree, until it reaches a leaf block. The advantage of the GHOST rule is that it guarantees the irreversibility of the selected pivot chain even when facing forks of honest nodes due to network delays as the blocks

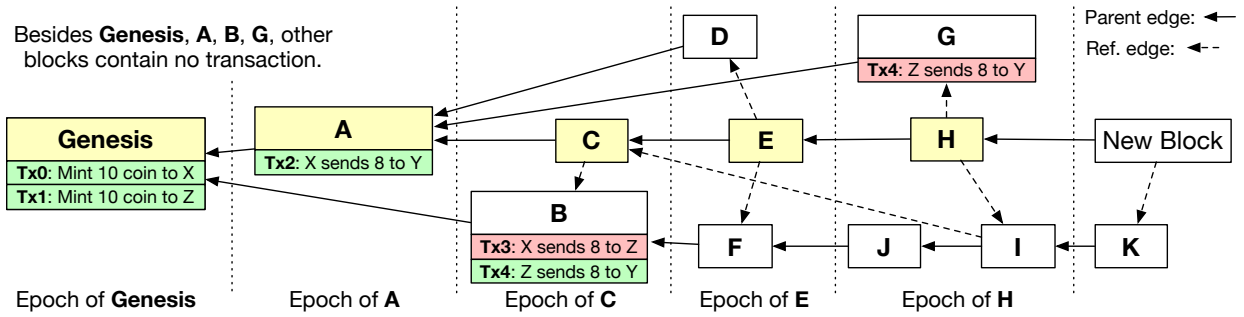

Figure 2: An example local DAG state to illustrate the consensus algorithm of Conflux. The yellow blocks are on the pivot chain in the DAG. Each block on the pivot chain forms a new epoch to partition blocks in the DAG.

in the forks also contribute to the safety of the pivot chain according to the analysis [26].

In Figure 2, Conflux selects Genesis, A, C, E, and H as the pivot chain. Note that this is not the longest chain in the parental tree and the longest chain is Genesis, B, F, J, I, and K. Conflux does not select this longest chain because the subtree of A contains more blocks than the subtree of B. Therefore, the chain selection algorithm selects A over B at its first step.

Generating New Block: Whenever a node generates a new block, it first computes the pivot chain in its local DAG state and sets the last block in the chain as the parent of the new block. This makes the chain heavier (i.e., the corresponding subtree of each block on the chain contains one extra block), so that other nodes will be even more likely to select the same chain as their pivot chains in future. The node then finds all tip blocks in the DAG that have no incoming edge and creates reference edges from the new block to each of those tip blocks. For example, if the node has a local state as shown in Figure 2, when generating a new block, the node will choose H as the parent of the new block and will create a reference edge from the new block to K.

Epoch: Parent edges, reference edges, and the pivot chain together enable Conflux to split all blocks in a DAG into epochs. As shown in Figure 2, every block in the pivot chain corresponds to one epoch. Each epoch contains all blocks 1) that are reachable from the corresponding block in the pivot chain via the combination of parent edges and reference edges and 2) that are not included in previous epochs. For example, in Figure 2, J belongs to the epoch of H because J is reachable from H but not reachable from the previous pivot chain block, E.

Block Total Order: Conflux determines the total order of the blocks in a DAG as follows. Conflux first sorts the blocks based on their corresponding epochs and then sorts the blocks in each epoch based on their topological order. If two blocks in an epoch have no partial order relationship, Conflux breaks ties deterministically with the unique ids of the two blocks. For the local DAG in Figure 2, Conflux obtains the total order as the following: Genesis, A, B, C, D, F, E, G, J, I, H, and K.

Transaction Total Order: Conflux first sorts transactions based on the total orders of their enclosing blocks. If two transactions belong to the same block, Conflux sorts the two transactions based on the appearance order in the block.

Conflux checks the conflicts of the transactions at the same time when deriving the orders. If two transactions are conflicting with each other, Conflux will discard the second one. If one transaction appears in multiple blocks, Conflux will only keep the first appearance and discard all redundant ones. In Figure 2, the transaction total order is Tx0, Tx1, Tx2, ~~Tx3~~, Tx4, and ~~Tx4~~. Conflux discards Tx3 because it conflicts with Tx2.

2.3 Security Analysis

We next discuss potential attack strategies and explain why Conflux is safe against these attacks. For a more formal discussion of the Conflux safety, see Section 3.3.

Now suppose an attacker wants to revert the transaction Tx4 in the block B in Figure 2. To do so, the attacker needs to revert the agreed total order of the blocks so that the attacker can insert an attacker block before B and that the attacker block contains a transaction which conflicts with Tx4. One naive attack strategy is to link such an attacker block to existing blocks in very early epochs, e.g., setting Genesis as the parent. However, because the attacker block is new with no children, it will not become a pivot chain block. By our epoch definition, the attacker block has to wait for the references of future pivot chain blocks. The block will still belong to a future epoch de-

spite the fact that it sets Genesis as its parent. Therefore the block will not appear before B in the total order.

Therefore to revert a transaction enclosed by a block, the attacker has to revert the partition scheme of epochs before the block. Because the epoch partition scheme is deterministically defined based on the pivot chain, the attacker therefore has to revert the corresponding blocks on the pivot chain. If the attacker attempts reverting the pivot chain, the situation is then similar to the double spending attack in chain-based Nakamoto consensus protocol. In this situation, the safety property of Conflux relies on the fact that all honest nodes will continue to work on the pivot chain to make the pivot chain longer and heavier. Since honest nodes together have more block generation power than the attacker, as the time passes by, early blocks on the pivot chain are increasingly irreversible for the attacker.

Transaction Confirmation: A user can confirm his or her transaction with the following strategy. The user locates the first epoch that contains a block including the transaction. The user identifies the corresponding pivot chain block of the epoch. The user then decides how much risk he or she can tolerate based on the estimations of the block generation power that the attacker controls. The user finally estimates the risk of the pivot chain block being reverted using the formula in Section 3.3 to decide whether to confirm the transaction.

3 Conflux Consensus

In this section we present the Conflux consensus algorithm and discuss its safety and liveness properties.

3.1 Consensus Algorithm

At any time, the local state of a user in the Conflux protocol is a graph $G = \langle B, g, P, E \rangle$. B is the set of blocks in G. $g \in B$ is the genesis block. P is a function that maps a block b to its parent block $P(b)$. Specially, $P(g) = \perp$. E is the set of directed reference edges and parent edges in this graph. $e = \langle b, b' \rangle \in E$ is an edge from the block b to the block b', which denotes that b' happens before b. Note that there is always a parent edge from a block to its parent block (i.e., $\forall b \in B, \langle b, P(b) \rangle \in E$). All nodes in the Conflux protocol share a predefined deterministic hash function Hash that maps each block in B to a unique integer id. It satisfies that $\forall b \neq b', \text{Hash}(b) \neq \text{Hash}(b')$.

We next define several utility functions and notations. Chain() returns the chain from the genesis block to a given block following only parent edges. Child() returns the set of child blocks of a given block. Sibling() returns the set of siblings of a given block. Subtree() returns the sub-tree of a given block in the parental tree. Before() returns the set of blocks that are immediately generated before a given block. Past() returns the set of blocks that are generated before a given block (but including the block itself). Figure 3 presents the definition of these utility functions. In the rest of this section, we use ordered lists to denote chains and serialized orders. "$\circ$" denotes the concatenation of two ordered lists.

$$G = \langle B, g, P, E \rangle$$

$$\text{Chain}(G,b) = \begin{cases} g & b = g \\ \text{Chain}(G, P(b)) \circ b & \text{otherwise} \end{cases}$$
$$\text{Child}(G,b) = \{b' \mid P(b') = b\}$$
$$\text{Sibling}(G,b) = \text{Child}(G, P(b)) - \{b\}$$
$$\text{Subtree}(G,b) = (\cup_{i \in \text{Child}(G,b)} \text{Subtree}(G,i)) \cup \{b\}$$
$$\text{Before}(G,b) = \{b' \mid b' \in B, \langle b', b \rangle \in E\}$$
$$\text{Past}(G,b) = (\cup_{i \in \text{Before}(G,b)} \text{Past}(G,i)) \cup \{b\}$$
$$\text{TotalOrder}(G) = \text{ConfluxOrder}(G, \text{Pivot}(G,g))$$

Figure 3: The Definitions of Chain(), Child(), Sibling(), Subtree(), Before(), Past(), and TotalOrder().

Input : The local state $G = \langle B, g, P, E \rangle$ and a starting block $b \in B$
Output: The last block in the pivot chain for the subtree of b in G

```
1  if Child(G,b) = ∅ then
2      return b
3  else
4      s ⟵ ⊥
5      w ⟵ −1
6      for b' ∈ Child(G,b) do
7          w' ⟵ |Subtree(G,b)|
8          if w' > w or w' = w and Hash(b') < Hash(s)
             then
9              w ⟵ w'
10             s ⟵ b'
11     return Pivot(G,s)
```

Figure 4: The definition of Pivot(G,b).

Pivot Chain Selection: Figure 4 presents our pivot chain selection algorithm (i.e., the definition of Pivot()). Given a DAG state G, Pivot(G,g) returns the last block in the pivot chain starting from the genesis block g. The algorithm recursively advances to the child block whose corresponding subtree has the largest number of blocks (lines 4-10). When there are multiple child blocks with the same number, the algorithm selects the child block

Local State: A graph $G = \langle B, g, P, E \rangle$

1 **while** *Node is running* **do**
2 **upon event** *Received* $G' = \langle B', g, P', E' \rangle$ **do**
3 $G'' \longleftarrow \langle B \cup B', g, P \cup P', E \cup E' \rangle$
4 **if** $G \neq G''$ **then**
5 $G \longleftarrow G''$
6 Broadcast the updated G to other nodes
7 **upon event** *Generated a new block b* **do**
8 $a \longleftarrow \text{Pivot}(G, g)$
9 $E' \longleftarrow E \cup \{\langle b, t \rangle \mid \forall b' \in B, \langle b', t \rangle \notin E\}$
10 $G \longleftarrow \langle B \cup \{b\}, g, P[b \mapsto a], E' \rangle$
11 Broadcast the updated G to other nodes

Figure 5: The Conflux Main Loop

Input : The local state $G = \langle B, g, P, E \rangle$ and a block a
Output: A list of blocks $L = b_1 \circ b_2 \circ \ldots \circ b_n$, where $b_1 = g$ and $\forall 1 \leq i \leq n, b_i \in B$

1 $a' \longleftarrow P(a)$
2 **if** $a' = \perp$ **then**
3 **return** a
4 $L \longleftarrow \text{ConfluxOrder}(G, a')$
5 $B_\Delta \longleftarrow \text{Past}(G, a) - \text{Past}(G, a')$
6 **while** $B_\Delta \neq \emptyset$ **do**
7 $G' \longleftarrow \langle B_\Delta, g, P, E \rangle$
8 $B'_\Delta \longleftarrow \{x \mid |\text{Before}(G', x)| = 0\}$
9 Sort all blocks in B'_Δ in order as $b'_1, b'_2, \ldots, b'_k$
such that $\forall 1 \leq i < j \leq k$, $\text{Hash}(b'_i) < \text{Hash}(b'_j)$
10 $L \longleftarrow L \circ b'_1 \circ b'_2 \circ \ldots \circ b'_k$
11 $B_\Delta \longleftarrow B_\Delta - B'_\Delta$
12 **return** L

Figure 6: The Definition of ConfluxOrder().

with the smallest unique hash id (line 8). The algorithm terminates until it reaches a leaf block (lines 1-2).

Consensus Main Loop: Figure 5 presents the main loop of a node running the consensus algorithm. It processes two kinds of events. The first kind of events is receiving DAG update information from other nodes via the underlying gossip network. The node updates its local state accordingly (lines 2-3) and relays the local state via the gossip network (lines 4-6). Note that here we assume that the underlying gossip network already verified the integrity of the received information (i.e., cryptographic signatures in the blocks).

The second kind of events corresponds to the local block generator successfully generating a new block b. In this case, the node adds b into its local DAG G and updates G as follows. It first sets the last block of the pivot chain in its local DAG as the parent of b (lines 8 and 10). The node also finds all of those blocks in the local DAG that have no incoming edges and creates a reference edge from b to each of those blocks (line 9). The node finally broadcast the updated G to other nodes via the gossip network. Note that $P[b \mapsto a]$ at line 10 denotes the result map of mapping b to a in P (other mappings in P are unchanged).

Note that for brevity, the pseudo-code in Figure 5 broadcasts the whole graph to the network. In our Conflux implementation, Conflux broadcasts and relays each individual block to avoid unnecessary network transmissions. See Section 4 for the details.

Total Order: Figure 6 defines ConfluxOrder(), which corresponds to our block ordering algorithm. Given the local state G and a block a in the pivot chain, ConfluxOrder(G, a) returns the ordered list of all blocks that appear in or before the epoch of a. Using ConfluxOrder(), the total order of a local state G is defined as TotalOrder(G) in Figure 3.

The algorithm in Figure 6 first recursively orders all blocks in previous epochs (i.e., the epoch of $P(a)$ and before). It then computes all blocks in the epoch of a as B_Δ (line 5). It topologically sorts all blocks in B_Δ and appends it into the result list (lines 6-12). The algorithm uses the unique hash id to break ties (lines 8-9).

3.2 Assumptions and Parameters

We next present the protocol assumptions that are important to our discussion of the safety and liveness properties of the consensus algorithm. Our protocol has the following assumptions and relevant parameters.

Block Generation Rate: The network together has a block generation rate of λ. We use λ_h to denote the combined block generation rate of honest nodes. We use $\lambda_\text{a} = q \cdot \lambda_\text{h}$ to denote the block generation rate of the attacker. Therefore $\lambda = \lambda_\text{h} + \lambda_\text{a}$ and $0 \leq q < 1$.

d-Synchronous: We assume the underlying gossip network provides d-synchronous communications for all honest nodes. If at time t, one honest node broadcast a block or transaction via the gossip network, then before time $t + d$, all honest nodes will receive this block and add this block into their local states.

Adversary Model: The attacker can choose arbitrary strategies to disrupt honest nodes. We assume two limitations to the capability of the attacker. Firstly, the attacker does not have the capability to reverse cryptographic functions. Therefore honest nodes can reliably verify the integrity of a block in the presence of the at-

tacker. Secondly, all honest nodes combined together have stronger block generation power than the attacker, i.e., as we noted before $\lambda_a = q \cdot \lambda_h$ and $0 \leq q < 1$.

When the attacker generates a new block, the attacker can create parent and reference edges from the new block to arbitrary existing blocks, not necessarily following our protocol. Note that because all blocks in Conflux is protected by cryptographic functions, the attacker however cannot modify edges associated with an already generated block even if the block is generated by the attacker himself/herself. The attacker can also withhold this new block for a certain period of time before broadcasting the block. Because Conflux implements a stale block detection mechanism similar to Bitcoin (see Section 4), the attacker cannot withhold blocks forever (but it can do so for hours). If the attacker decides to send a block to any honest node, due to our d-synchronous assumption, all honest nodes will see the block after a while.

3.3 Correctness

Safety: Conflux applies the GHOST rule to select its pivot chain. The pivot chain in Conflux therefore satisfies the following lemma (Proposition 2 in [26]):

Lemma 1 *Suppose b is a block generated at time t and b is on the pivot chain of all honest nodes at time $t+\tau$. The chance of b failing off the pivot chain after $t+\tau$ goes to zero as τ goes to infinity.*

Lemma 1 states that as long as the attacker controls less than half of the block generation power, early blocks on the pivot chain are stabilized. The rationale of Lemma 1 is that all honest nodes will eventually generate new blocks only under the subtree of b. Because the combined block generation rate of all honest nodes is greater than the attacker, by the large number law, after waiting a sufficiently long period of time it will be impossible for the attacker to make a subtree without b heavier than the subtree of b. See [26] for the proof of this lemma.

For a pivot chain block b, we define the b-prefix order as the prefix of the block total order that ends with b. In Conflux, transactions in b are irreversible, if the b-prefix orders of all honest nodes no longer change. We have the following theorem for the irreversibility of the b-prefix order.

Theorem 2 *Suppose G is the local DAG of an honest node at time t, all blocks in* $\text{Chain}(G,b)$ *are generated before t,* $\text{Chain}(G,b)$ *is the common prefix of the pivot chains of all honest nodes at time $t+\tau$. Then 1) the b-prefix orders of all honest nodes agree with each order at time $t+\tau$ and 2) the chance of the b-prefix order of any honest node changing after $t+\tau$ goes zero as τ goes to infinity.*

Proof. Because $\text{Chain}(G,b)$ is the common prefix of the pivot chains of all honest nodes and the algorithm in Figure 6 generates a common total order prefix given a common pivot chain prefix, so 1) holds. Applying Lemma 1 to each block in $\text{Chain}(G,b)$, we know that the chance of $\text{Chain}(G,b)$ no longer being the common prefix of the pivot chains of all honest nodes after $t+\tau$ goes to zero as τ goes to infinity. Again because the algorithm in Figure 6 generates the same total order prefix if the pivot chain prefix does not change, the chance of the algorithm producing a different b-prefix order goes to zero as τ goes to infinity. □

Liveness: As discussed above, the consensus of the block total order prefix in Conflux depends on the consensus of the pivot chain prefix. We have the following theorem to guarantee that there will always be new blocks appended to the common prefix of the pivot chain.

Theorem 3 *At any time t, $\exists \alpha < \infty$ such that at least one new block will be added to the common prefix of the pivot chains of all honest nodes at time $t+\alpha$.*

Proof. We consider a special event at $t+\alpha$ such that 1) there is at least one block generated during $[t, t+\alpha-(c+1)\cdot d)$ and 2) the whole system does not generate any block during $[t+\alpha-(c+1)\cdot d, t+\alpha)$, where c is the number of blocks the attacker withheld at the start of this time period. Because the attacker can only withhold generated blocks for a bounded period of time (e.g., hours), c is bounded. Therefore this event will eventually occur. Once this event occurred, all honest nodes would see the same DAG states. Because the algorithm in Figure 4 is deterministic, all honest nodes would therefore select the same pivot chain and the chain would include at least one new block generated during $[t, t+\alpha-(c+1)\cdot d)$. □

Confirmation: For any block b' in a DAG, suppose b' belongs to the epoch of b, where b is a pivot chain block. We can confirm the transactions in b' as long as b becomes irreversible on the pivot chain. Like standard Nakamoto consensus, although we can wait sufficiently long to make the risk of an attacker reverting b arbitrarily low. The risk will always be greater than zero. We have the following lemma and theorem to bound the risk of confirming b on the pivot chain:

Lemma 4 *Suppose b is a block on the pivot chain of all honest nodes during the time $[t-d, t]$ and $P(b)$, the parent of b, is generated at time zero. The chance of b being*

kicked out of the pivot chain by one of its sibling blocks a is no more than:

$$\sum_{k=0}^{n-m} \zeta_k \cdot q^{n-m-k+1} + \sum_{k=n-m+1}^{\infty} \zeta_k$$

where n is the number of blocks in the subtree of b before time $t-d$, m is the number of blocks in the subtree of a generated by honest nodes, and $\zeta_k = e^{-q\lambda_h t} \frac{(q\lambda_h t)^k}{k!}$.

Theorem 5 *Suppose b is a block on the pivot chains of all honest nodes during the time $[t-d,t]$. The chance of b falls off the pivot chain is no more than:*

$$\max_{\substack{a \in \text{Chain}(G,b) \\ a' \in \text{Sibling}(a)}} \Pr[a \text{ is kicked out of the pivot chain by } a']$$

Lemma 4 is a direct application of Theorem 10 in [26]. It provides us a way to estimate the stability of each individual block on the pivot chain. Theorem 5 indicates that the stability of a pivot chain prefix is determined by the least stable block in the prefix. See Appendix A for the proof of Lemma 4 and Theorem 5.

4 Implementation

We have implemented both Conflux and GHOST (for comparison in our evaluation) based on the Bitcoin Core codebase v0.16.0 [1].

Block Header: To implement reference edges in Conflux, we modified the block header structure in Bitcoin to include 32-byte block header hashes for each of its outgoing reference edges. Our experimental results show that this introduces a negligible overhead of less than 960 bytes per block. Note that if a proof-of-work (PoW) scheme is used to generate new blocks, these reference hashes must be included as part of the puzzle to avoid having attackers be able to generate blocks with different references at essentially zero-cost.

Gossip Network: Both Conflux and GHOST require to maintain the full structure of the tree/DAG of blocks, while Bitcoin Core only propagates blocks in the identified longest chain. We therefore modified the gossip network layer of Bitcoin Core to broadcast and relay all blocks. To ensure that when a block is delivered to the consensus layer, all of its past blocks are already delivered, Conflux maintains the validity for each block. Whenever Conflux receives a block, it traverses the DAG structure using breadth-first search (BFS) and updates the validity of each traversed block. A block is valid if and only if Conflux has received all past blocks (i.e., blocks that are reachable via parent and reference edges) of the block. Conflux then delivers all of the newly validated blocks to the consensus layer.

Detecting Stale Blocks: Bitcoin Core has the following mechanism to detect stale blocks (e.g., blocks generated and withheld by attackers). Each node periodically synchronizes with its peers to maintain a network-adjusted time, which is the median of the timestamps returned by its peers. Each block in Bitcoin Core is also timestamped. A new block will be flagged as invalid if the timestamp of the new block is earlier than the median timestamp of previous 11 blocks or if the timestamp of the new block is two hours later than the network-adjusted time.

We use the same mechanism in Bitcoin Core with the following modifications. First, we modify the rule proportionally to the block generation rates which we use in our experiments. For example, if the system generates a block every 20 seconds (i.e., 30 times faster than Bitcoin), then a block is considered invalid if its timestamp is earlier than the median timestamp of previous 330 blocks. Secondly, Conflux does not delete blocks with invalid timestamps. It simply ignores this invalid block when counting the number of blocks in a subtree for selecting the pivot chain. The rationale is that as long as the invalid block no longer affects the partition scheme of already stable epochs, it is safe to include the block into future epochs, processing transactions inside the block.

Bootstrapping a Node: When a node starts, it will handshake with each of its peers and run a bootstrapping one-way synchronization process to update its local state. For both GHOST and Conflux, the node needs to download all blocks in the tree/DAG from its peers, not just the selected chain. To implement this one-way synchronization, we enhanced the Bitcoin Core codebase with four extra message types: `gettips`, `tips`, `getchains` and `chains`. For sake of simplicity, let us consider the case where a node *A* attempts to download the blocks from another node *B*. *A* first sends *B* a `gettips` message requesting the list of tips, i.e., leaf blocks in the (parental) tree (represented by their 32-byte hashes). `tips` is used in response to `gettips` to retrieve the block hash of all the tips *B* is aware of. For each of *B*'s tip, *A* checks whether this is an unknown block or not and packs all the answers in a `getchains` message. Upon receiving this `getchains` message, for each new tip to *A*, *B* computes the last known block in the chain from the genesis block to this tip and sends *A* a `chain` message containing the list of blocks starting right after the last known block.

5 Experimental Results

We next present a quantitative evaluation of Conflux to answer the following questions:

1. What is the throughput that Conflux can achieve for obtaining the block total order?
2. What is the confirmation time that a user must wait in Conflux for obtaining high confidence of irreversibility? How does this confirmation time correlate with the block generation power that the attacker controls and the risk that the user is willing to tolerate?
3. How does Conflux compare to previous chain-based Nakamoto consensus protocols like Bitcoin [20] and GHOST [26]?
4. How does Conflux scale as the network bandwidth changes? How does Conflux scale as the number of full nodes grow?

We deployed Conflux on up to 800 Amazon EC2 m4.2xlarge virtual machines (VM), each of which has 8 cores and 1Gbps network throughput. By default, we run 25 Conflux full nodes in each VM and limit the bandwidth of each full node to 20Mbps. To model the network latency, we use the inter-city latency measurements [27] and assign each VM to one of 20 major cities. We simulate the inter city delay by inserting artificial delays before the message delivery. For each full node, the gossip network of Conflux connects it to an average of 10 randomly selected peers. In our experiments, we assign all full nodes with an equal block generation power. For each generated block, we use the testing utilities in Bitcoin core code base to fill the block full with artificial transactions. To avoid unnecessary PoW computation, we simulate the mining with a poisson process.

We then deployed Bitcoin and our implementation of GHOST under the same setup as Conflux. We run up to 20k full nodes in our experiments. At April 2018, Bitcoin has fewer than 12k full nodes [2] and Ethereum [6] has fewer than 17k full nodes. Our experiments are at the same scale as those real-world cryptocurrencies.

Note that in our experiments we measure the network diameter d as the propagation time for 99% of the blocks to reach all full nodes. This enables us to see the trend of network diameter (see Section 5.3). There is always a few node generating blocks when they are lagging behind, causing longer delays. Note that this phenomena does not affect the correctness of Conflux because we can simply count those nodes that lags behind as malicious nodes. Also note that For all the experiments, we monitored the overhead introduced by Conflux due to its DAG-based approach. The computation overhead is negligible compared to the PoW puzzles, and the spatial overhead is a maximum of 960 bytes per block, which is small compared to typical block size in several MBs.

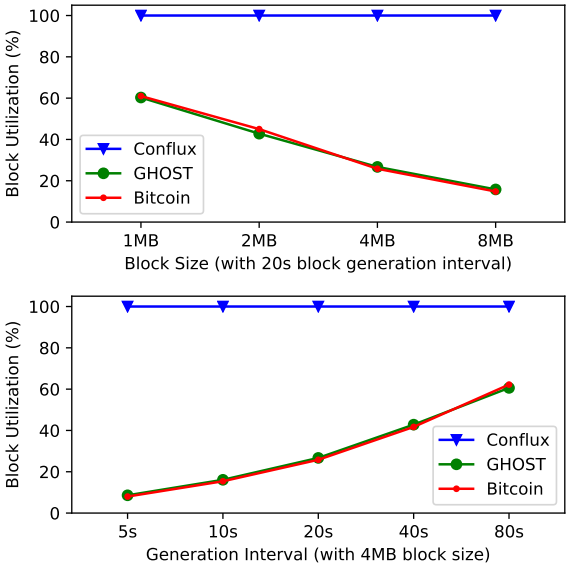

Figure 7: Block utilization ratio.

5.1 Throughput

To evaluate the throughput improvement, we run Conflux, Bitcoin, and GHOST with 10k full nodes (i.e., 400 VMs) under the following two configurations: 1) increasing the block size limit from 1MB to 8MB with fixed block generation rate at 20 seconds per block; 2) decreasing the block generation rate from 5 seconds per block to 80 seconds per block with fixed block size limit at 4MB. We run each protocol for two hours for each configuration.

Figure 7 presents the results, where X-axis corresponds to different configuration settings and Y-axis tells the correspondent block utilization ratio. For Bitcoin and GHOST, this ratio corresponds to the portion of blocks in the selected chain. For Conflux, this number is always 1 because all blocks will be eventually included. Note that the consensus protocol throughput is the multiplication of three numbers: the block size limit, the block generation rate, and the block utilization ratio.

The results show that Conflux achieves a throughput of 2.88GB/h under the block generation setting 4MB/5s. If we assume the same transaction size as the real-world

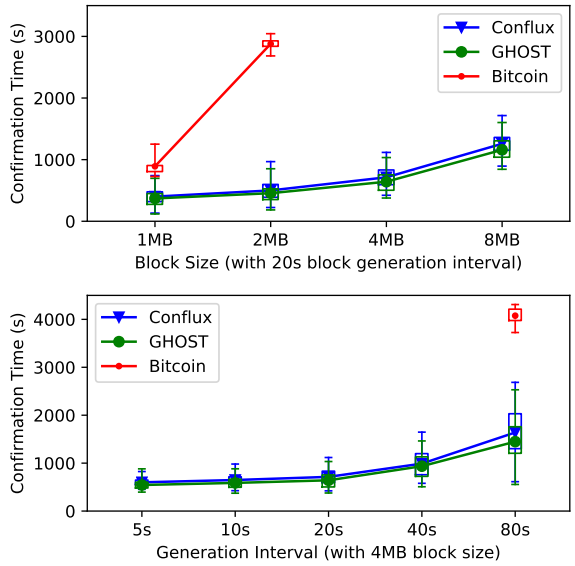

Figure 8: Confirmation time.

Bitcoin network, Conflux could process 3200 transactions per second. In fact, our results indicate that the throughput of Conflux is only limited by the processing capability of each individual node, i.e., Conflux can achieve even higher throughput if we lift the bandwidth limit to 40Mbps (See Section 5.3).

Our results also tell that as the block size and the block generation rate increase, more blocks are generated in parallel. For Bitcoin and GHOST, this indicates an increasing number of forks in the resulting block trees. For example, under the block generation setting 4MB/5s, only 8% and 8.6% of blocks are on the agreed chains of Bitcoin and GHOST, respectively. Blocks in forks will not be included in the result total order and resources are wasted for generating those blocks. Unlike Bitcoin and GHOST, Conflux is capable of processing all blocks. Conflux therefore achieves significantly higher throughputs than Bitcoin and GHOST, especially when the block size is large or the block generation rate is fast.

5.2 Confirmation Time

Figure 8 presents the average confirmation time of Conflux, Bitcoin, and GHOST under different configurations as in the same experiments above. Confirmation time is the time duration that a user has to wait to obtain a high confidence that the total order of a block will not change (i.e., the prefix of the total order before this block does not change). In this setting, the user confirms a block if the attacker has less than 0.01% chance to revert its transaction, assuming the attacker controls less than 20% of the network block generation power (i.e., $q < 0.25$). The error bar in Figure 8 shows the medium, 25%, 75%, minimum, and maximum confirmation time of blocks for each protocol and each configuration.

Our results show that Conflux can confirm blocks in minutes. When using the block generation setting of 4MB/5s, Conflux confirms blocks in 10.0min on average. Under all configurations, users in Conflux wait for a similar confirmation time as GHOST. This is expected, because the confirmation of Conflux blocks relies on the confirmation of the corresponding pivot chain blocks which follows the same GHOST rule. Note that under all settings except 1M/20s, 2M/20s, and 4M/80s, Bitcoin is unable to confirm any block because the ratio of the longest chain is too small against any attacker with 20% of the network block generation power.

The results also show that as the block size increases, blocks take longer to get confirmed on all three protocols. This is because as explained in Figure 7, using larger blocks will cause more blocks generated in parallel. Some nodes may temporarily generate blocks that are not under the last block of the chain (or the pivot chain in Conflux).

Our results further show that increasing the block generation will grow the chain (or the pivot chain in Conflux) faster and therefore confirm blocks faster. But this effect diminishes as the block generation rate approaches the processing capability of individual nodes, because frequent concurrent blocks and forks will slow down the confirmation.

Attacker Capability and Confidence Ratio: The top plot in Figure 9 shows, under the block generation setting of 4M/10s, how the confirmation time changes for Conflux and GHOST, if the user assumes different attacker capability. Note that Bitcoin cannot confirm confirmations in this setting. Our results show that even the user assumes attackers controlling 30% of the block generation power, Conflux can still confirm blocks in a medium of 16.8 minutes with confidence 99.99%. As the attacker controls more power, the confirmation time grows exponentially for both protocols. The bottom plot shows, under the setting of 4M/10s, as the user waits longer how the confirmation risk changes for Conflux, Bitcoin, and GHOST. Our results show that the chance of attackers reverting the total order prefix of a confirmed block drops exponentially as the user waits longer.

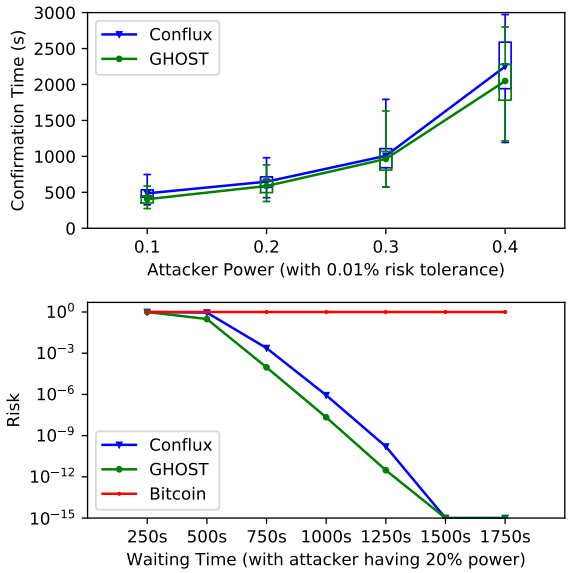

Figure 9: Risk tolerance.

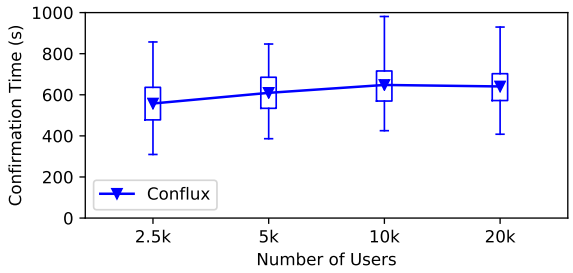

Figure 10: Confirmation time of Conflux with 4MB block size, 10s generation interval, and 2.5k to 20k users

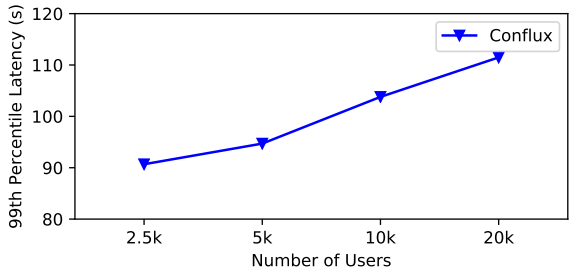

Figure 11: Network diameter d of Conflux with 4MB block size, 10s generation interval, and 2.5k to 20k users

5.3 Scalability

To study the scalability of Conflux, we run Conflux with the following two configurations: 1) increasing the number of deployed full nodes from 2.5k to 20k with block size limit at 4MB and block generation rate at 10s per block (4MB/10s); 2) lifting the bandwidth limit to 40Mbps per node with block size limit at 4MB and block generation rate at 2.5s per block (4MB/2.5s).

Scalability with Higher Bandwidth Limit: In our experiments, we found that the throughput bottleneck of Conflux becomes the processing capability of each individual node, especially the bandwidth limit. Conflux cannot run under the block generation setting of 4M/2.5s simply because the gossip network does not have enough bandwidth to propagate blocks under this fast rate. We then lifted the bandwidth limit of each node from 20Mbps to 40Mbps and run Conflux on 10k full nodes with the setting of 4M/2.5s again. Conflux runs successfully and achieves the throughput of 5.76G/h. In this run, Conflux also confirms blocks in 5.68 minutes on average. If we assume the same transaction size as the real-world Bitcoin network, Conflux would process 6400 transactions per second.

Scalability with More Nodes: Figure 10 shows the confirmation time of Conflux with 2.5k, 5k, 10k, and 20k full nodes under the setting of 4MB/10s. X-axis corresponds to the number of full nodes while Y-axis tells the confirmation time in seconds. Figure 11 shows the network diameter of Conflux with 2.5k, 5k, 10k, and 20k full nodes under the setting of 4M/10s. X-axis corresponds to the number of full nodes while Y-axis tells the network diameter d (for propagating 99% of blocks). Our results show that Conflux scales well to 20k full nodes and achieves average confirmation time under 10.7 minutes. Note that the achieved throughput of the 4MB/10s setting is always 720MB/h. Our results also show that the network diameter grows linearly as the number of full nodes doubling. The results show that even with 20k users, the increment of the network diameter is small. Therefore the confirmation time of of a Conflux transaction is still dominated by waiting enough blocks building on top the corresponding pivot chain block that processes the transaction. Because we are using the same 4MB/10s setting, this waiting time stays mostly the same so does the confirmation time.

6 Related Work

DAG-based consensus: People have proposed several consensus protocols over DAG-based blockchains. SPECTRE [24] specifies a non-transitive partial orders for all pairs of blocks in the DAG, while Conflux pro-

vides a total order over all transactions which is critical to support applications like smart contracts.

Inclusive blockchains [17] extends the Nakamoto consensus to DAG and specifies a framework to include off-chain transactions in a consistent manner. PHANTOM [25] shares the same aspects of Conflux in terms of specifying a total order across transactions in DAG-based blockchains. In PHANTOM, participating nodes first find an approximate k-cluster solution for its local block DAG to prune potentially malicious blocks, then topologically sorts the remaining blocks to obtain a total order. PHANTOM, however, is vulnerable to liveness attacks. Attackers with little computation power can delay the confirmation of transactions indefinitely with high probabilities even all honest nodes are completely synchronous. See Appendix B for the attack.

Besides the aforementioned differences, to our best knowledge Conflux presents the first empirical evaluation of DAG-based blockchains. Running 10k full nodes on EC2 where each full node has 40Mbps of bandwidth, Conflux commits 5.76GB of transactions per hour and confirms them within 4.5-7.4 minutes. There is no empirical evaluation of other DAG-based protocols and it is therefore unclear what is the throughput and the confirmation time of these protocols once implemented and deployed.

Nakamoto consensus: The Nakamoto consensus protocol and the GHOST rule specify how the nodes should choose a single canonical chain when encountering multiple forks [20, 26]. The end result is that all honest nodes converge on the canonical chain on a high probability. The canonical chain corresponds to a total-ordered, irreversible log of transactions, where blocks and transactions that are not on the canonical chains are discarded and do not contribute to the throughput.

Instead of choosing one single canonical chain, Conflux assigns a total order of non-conflicting transactions over the DAG. In Conflux, blocks that are not on the pivot chains also contribute to the throughput while Conflux still maintains a total-ordered, irreversible log of transactions for the users, resulting in significant performance boost compared to Bitcoin and GHOST.

Consortium consensus: Much research explorers the direction of reducing the uses of the expensive Nakamoto consensus in blockchains to improve their performances. Bitcoin-NG [10] elects a leader using the Nakamoto consensus protocol and the leader is responsible to commit all transactions until the next leader is elected. Several research work has proposed to combine Nakamoto consensus with BFT protocols [16, 23], or to fully replace Nakamoto consensus with BFT protocols [12, 18, 19].

From a practical point of view, all the proposals above run the alternative consensus protocols within a confined group (one node for Bitcoin-NG) of nodes, since protocols like BFT only scale up to dozens of nodes in practices. Therefore one key challenge of these systems need to address is to choose the confined group in an adversarial environment like blockchains while maintaining the security guarantees. For example, the groups can be chosen based on their stakes of the system [12] or external hierarchy of trusts [18].

Conflux differs from the above approaches in two ways. First, the total orders of the transactions is decided by all participants of the network instead of a confined group. Additionally Conflux is able to tolerate to half of the network are malicious while the BFT-based approaches can only tolerate up to one third of malicious nodes. Second, the above approaches enforce the total order *eagerly* as the members of the confined group fully verify and commit the transactions before moving on to the next ones. Conflux, however, allows multiple blocks generating in parallel and finalizes their orders later. The design decision presents an interesting trade-off between throughput and latency in the system. For example, Conflux achieves 3.84x throughput compared to Algorand, but the confirmation time in Algorand is shorter than Conflux.

Fairness: Recent studies have shown that large miners with more than 25% of computational power can capture unproportionally more rewards, putting smaller miners in disadvantages [11, 21, 22]. Although achieving fairness is outside the scope of this paper, we note that by adapting faster block generation rates and allowing multiple blocks generated in parallel, Conflux inherently mitigates the disadvantages of small miners. It is not possible for a large miner to invalidate blocks generated by small miners via forking the chain.

Proof-of-Stake: The original Nakamoto consensus protocol in Bitcoin requires nodes to solve significant computation puzzles (i.e., proof-of-work (PoW)) to vote for consensus. As the PoW scheme demands a significant amount of resources, alternative schemes such as proof-of-stake (PoS) has been proposed [7, 14, 15]. In PoS based system the leader is elected based upon the stakes he or she owns in the system. The leader then is responsible to append new blocks to the blockchain. Conflux is complementary to the PoS scheme. The consensus algorithm can be adopted by the PoS-based blockchains as long as the PoS mechanisms can maintain a stable network block generation rate.

7 Conclusion

Conflux is a fast, scalable, and decentralized blockchain platform with proved safety. It exploits the inherent parallelism among blockchain transactions, uses a DAG-based approach to defer the total order reconciliation while providing the externally same interface compared to traditional chain-based approaches. It provides orders of magnitude throughput improvement, as validated by the real deployment in Amazon EC2 clusters. Conflux provides a promising solution to address the performance bottleneck of blockchains and opens up a wide range of blockchain applications.

Acknowledgement

We thank Zhenyu Guo and Haohui Mai for valuable feedbacks on early drafts of this paper. We thank Guang Yang and Jialin Zhang for the help on the consensus algorithm.

References

[1] Bitcoin Core. https://github.com/bitcoin/bitcoin.

[2] Bitnodes. https://bitnodes.earn.com/.

[3] CryptoKitties. https://www.cryptokitties.co/.

[4] Etherum White Paper. https://github.com/ethereum/wiki/wiki/White-Paper.

[5] Skyrocketing fees are fundamentally changing bitcoin. https://arstechnica.com/tech-policy/2017/12/bitcoin-fees-rising-high/.

[6] The ethereum node explorer. https://www.ethernodes.org/.

[7] BUTERIN, V., AND GRIFFITH, V. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437* (2017).

[8] DELOITTE. 5 blockchain technology use cases in financial services. https://www2.deloitte.com/nl/nl/pages/financial-services/articles/5-blockchain-use-cases-in-financial-services.html.

[9] DELOITTE. Blockchain: Opportunities for health care. https://www2.deloitte.com/us/en/pages/public-sector/articles/blockchain-opportunities-for-health-care.html, 2018.

[10] EYAL, I., GENCER, A. E., SIRER, E. G., AND VAN RENESSE, R. Bitcoin-ng: A scalable blockchain protocol. In *NSDI* (2016), pp. 45–59.

[11] EYAL, I., AND SIRER, E. G. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security* (2014), Springer, pp. 436–454.

[12] GILAD, Y., HEMO, R., MICALI, S., VLACHOS, G., AND ZELDOVICH, N. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), ACM, pp. 51–68.

[13] IBM. Blockchain for supply chain. https://www.ibm.com/blockchain/supply-chain/.

[14] KIAYIAS, A., RUSSELL, A., DAVID, B., AND OLIYNYKOV, R. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference* (2017), Springer, pp. 357–388.

[15] KING, S., AND NADAL, S. Ppcoin: peer-to-peer crypto-currency with proof-of-stake (2012). *https://peercoin.net/assets/paper/peercoin-paper.pdf*. (2017).

[16] KOGIAS, E. K., JOVANOVIC, P., GAILLY, N., KHOFFI, I., GASSER, L., AND FORD, B. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)* (2016), pp. 279–296.

[17] LEWENBERG, Y., SOMPOLINSKY, Y., AND ZOHAR, A. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security* (2015), Springer, pp. 528–547.

[18] MAZIERES, D. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation* (2015).

[19] MILLER, A., XIA, Y., CROMAN, K., SHI, E., AND SONG, D. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 31–42.

[20] NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf.

[21] NAYAK, K., KUMAR, S., MILLER, A., AND SHI, E. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *IEEE European Symposium on Security and Privacy* (2016), pp. 305–320.

[22] PASS, R., AND SHI, E. Fruitchains: A fair blockchain. In *Proceedings of the ACM Symposium on Principles of Distributed Computing* (2017), ACM, pp. 315–324.

[23] PASS, R., AND SHI, E. Hybrid consensus: Efficient consensus in the permissionless model. In *LIPIcs-Leibniz International Proceedings in Informatics* (2017), vol. 91, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[24] SOMPOLINSKY, Y., LEWENBERG, Y., AND ZOHAR, A. Spectre: Serialization of proof-of-work events: confirming transactions via recursive elections, 2016.

[25] SOMPOLINSKY, Y., AND ZOHAR, A. Phantom, a scalable blockdag protocol. https://eprint.iacr.org/2018/104.pdf.

[26] SOMPOLINSKY, Y., AND ZOHAR, A. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security* (2015), Springer, pp. 507–527.

[27] WONDERNETWORK. Global ping statistics: Ping times between WonderNetwork servers. https://wondernetwork.com/pings, Apr. 2018.

A Proof of correctness

A.1 Proof of lemma 4

Lemma 4 *Suppose b is a block on the pivot chain of all honest nodes during the time $[t-d,t]$ and $P(b)$, the parent of b, is generated at time zero. The chance of b being kicked out of the pivot chain by one of its sibling block a is:*

$$\sum_{k=0}^{n-m} \zeta_k \cdot q^{n-m-k+1} + \sum_{k=n-m+1}^{\infty} \zeta_k$$

where n is the number of blocks in the subtree of b before time $t-d$, m is the number of blocks in the subtree of a generated by honest nodes, and $\zeta_k = e^{-q\lambda_h t}\frac{(q\lambda_h t)^k}{k!}$.

Proof. We use G_t^v to denote the local state of participant v at global time t. G_t^∞ is the union set of the local state of all the participants (including attacker). G_t^{honest} is the union set of all the honest participants' local state.

While all honest participants are acknowledging b as pivot block, the expected waiting time for the next strengthening subtree of b is $1/\lambda_h$.

Now consider an honest participant in the worst network case: every blocks in the subtree of b have the maximum delay d, but the blocks in other subtrees have no delay. If a haven't kicked out of b in this participant, it won't revert b at any honest node.

In the node with the worst network case, the gap between pivot block b and its sibling a are $X_0 = |\text{Desc}(G_{t-d}^\infty, b)| - |\text{Desc}(G_t^{\text{honest}}, a)|$. According to theorem 10 in [26], attacker can extend a to be weighter than b with probability less than $\left(\frac{q\lambda_h}{\lambda_h}\right)^{X_0+1}$.

Let $m = \text{Desc}(G_t^{\text{honest}}, a)$, $n = \text{Desc}(G_{t-d}^\infty, b)$, k be number of blocks generated later than time zero. Then we can claim $|\text{Desc}(G_t^\infty, a)| \le k+m$, $X_0 \ge n-m-k$. The probability of generate k attacker blocks in time t is $\zeta_k = e^{-q\lambda_h t}\frac{(q\lambda_h t)^k}{k!}$, the chance of b being kicked out of the pivot chain by its sibling a is less than

$$\sum_{k=0}^{\infty} \zeta_k \cdot \min\left\{q^{n-m-k+1}, 1\right\}.$$

□

A.2 Proof of theorem 5

Theorem 5 *Suppose b is a block on the pivot chains of all honest nodes during the time $[t-d,t]$. The chance of b falls off the pivot chain is less than:*

$$\max_{\substack{a\in\text{Chain}(G,b)\\ a'\in\text{Sibling}(a)}} \Pr[a \text{ is kicked out of the pivot chain by } a']$$

Proof. For any time $t' > t$, let $m(t')$ be the blocks generated by attacker in $[t,t']$, $n(t')$ be the blocks generated by honest participants in $[t,t'-d]$. If the attacker reverts pivot block $\bar{a} \in \text{Past}(G,b)$ by $\bar{a}' \in \text{Sibling}(\bar{a})$ at time t'. We can claim

$$\begin{aligned} & m(t') - n(t') \\ >& |\text{Desc}(G_t^\infty, \bar{a})| - |\text{Desc}(G_t^\infty, \bar{a}')| \\ \ge& \min_{\substack{a\in\text{Chain}(G,b)\\ a'\in\text{Sibling}(a)}} |\text{Desc}(G_t^\infty, a)| - |\text{Desc}(G_t^\infty, a')| \end{aligned}$$

We use k denote

$$\min_{\substack{a\in\text{Chain}(G,b)|\\ a'\in\text{Sibling}(a)}} \text{Desc}(G_t^\infty, a)| - |\text{Desc}(G_t^\infty, a')|.$$

So $m(t') - n(t') > k$ is the necessary condition for a successful attack under any strategies.

If the attacker chooses $\tilde{a}, \tilde{a}'$ which minimize k and mines blocks under $\tilde{a}'$, $m(t') - n(t') > k$ is also the sufficient condition. Since $m(t')$ and $n(t')$ are independent with strategies, mining blocks under $\tilde{a}'$ is the dominant strategy for attacker. So the successful probability is dominate by the probability of reverting the most competitive $\tilde{a}'$. □

Remark: The probability space of lemma 4 and theorem 5 are different. In lemma 4, we estimate the probability given subtree size m and n. In theorem 5, we estimate the probability given all the blocks generated before time t.

B Attack on PHANTOM

B.1 Overview

PHANTOM [25] is a DAG-based protocol that attempts to achieve consensus on a total order of blocks. In PHANTOM, participants topologically sorts their local DAG. The algorithm guarantees results consistency among different participants and robustness of accepted transactions.

The topological sorting algorithm consists of two phases, in the first phase, the algorithm 2-colors all the blocks into blue and red to eliminate the potentially malicious blocks. Given a graph G, this phase contains 4 steps:

- For each tip (the blocks without decedent) $b \in tips(G)$, let $past(b)$ contains all the ancestors of b, coloring the subgraph $past(b)$ recursively.
- Let $|BLUE_k(G)|$ denote the number of blue blocks in 2-coloring result of G. Find the tip b_{max} which maximizes $|BLUE_k(past(b_{max}))|$.

- In graph G, color blocks in $past(b_{max})$ according to the result of subgraph $past(b_{max})$, color b_{max} in blue.

- Let anti(b) denotes the blocks which aren't the ancestors nor the decedents of b. For the blocks $b \in$ anti(b_{max}), color it in blue if anti(b) contains less than k blue blocks in G.

The score of block b is defined by $|BLUE_k(past(b))|$. A main chain is derived from this step. b_{max} is the chain tip. The highest scoring tip in $past(b_{max})$ is its predecessor in the chain, and so on.

In the second step, participants topological sorts all the blue blocks based on the main chain. In correctness proof of PHANTOM, robustness of topological order is based on robustness of its main chain.

B.2 Liveness attack

Here we show an attack for Algorithm 1 in PHANTOM[25]. This attack allows attacker to kick out a block from main chain arbitrarily late. Even if the attacker can only withhold finite blocks and has small block generation capability, it is able to kick out one block of main chain even this block has been accepted by all the honest node for arbitrary long time. This attack shows that PHANTOM cannot achieve safety and liveness at the same time.

Notation: In the following, we call the blocks generated by honest participants *honest blocks* and the the blocks generated by attacker *malicious blocks*. For any block c, anti(c) contains all the blocks which are not the ancestors nor the descendants of c, which is called *anti-set*. past(c) contains all the ancestors of c.

The attacker chooses an honest block as the start point, which is denoted by b_1. The honest blocks which refer b_1 as ancestor are denoted by $b_2, b_3, \cdots$ in time sequential. Also, we use $a_1, a_2, \cdots$ to denote the malicious blocks. Let B contain all the honest block, A contain all the malicious block.

Network Assumption: Here we use a weaker network assumption. We do not assume the attacker can control or delay the communication between honest participants. We instead assume all messages between any two nodes are delivered immediately, i.e., we assume a fully synchronous network [1].

We suppose $|B \cap \text{anti}(b_j)|$ has an upper bound k' for all the $b_j \in B$ with negligible exception. In real world, most mining computation power are in mining pools with good network synchronization. So this assumption is also reasonable without attacker.

We also assume b_1 is on the main chain of every honest blocks.

Parameter Assumption: Suppose the gap between upper bound k' and PHANTOM protocol parameter k is $k_\Delta = k - k'$. When all the blocks suffer a maximum network delay, the choice of k guarantees that $|\text{anti}(b_j)| \le k$ holds for almost all the honest block b_j. So in a high block generation rate, we can assume that k is large enough. Precisely,

$$k_\Delta(k_\Delta - 5) \ge 4k'.$$

Attack strategy: We define an positive integer array $\{h_i\}_{i=1}^{\infty}$ as following:

$$h_1 = 1,\ h_{n+1} = h_n + n - 1.$$

For each malicious block a_i, it refers $b_1 \sim b_{h_i}$ and $a_1 \sim a_{i-1}$ as its ancestors. Attacker withholds a_i after its generation. When $b_{h_{i-1+k_\Delta}}$ is generated, attacker made everyone receive a_i and $b_{h_{i-1+k_\Delta}}$ immediately.

Attacker can start to mine a_i when b_{h_i} has been generated. If $b_{h_{i-1+k_\Delta}}$ is generated earlier than a_i, the liveness attack fails.

Analysis: According to the previous strategy, every malicious blocks has a large anti-set and every honest blocks has a small anti-set.

Lemma 6 *For any $b_j \in B$, $|\text{anti}(b_j) \cap A| < k_\Delta$.*

Lemma 7 *For any $a_i \in A$, $|\text{anti}(a_i) \cap B| > k + k'$.*

These properties provides malicious block advantage in calculate blue set.

Lemma 8 *According to Algorithm 1 in PHANTOM, for any $b_j \in B$, $Blue_k(past(b_j)) \cap A = \emptyset$. For any $a_i \in A$, $Blue_k(past(a_i)) \cap B = past(a_i) \cap B$.*

Proof. Without loss of generality, we ignore the common ancestor $past(b_1)$ and regard b_1 as genesis block. We prove this lemma by induction.

This lemma holds for b_1 trivially since anti$b_1 = \emptyset$.

If this lemma holds for all the blocks in past(a_i), $|Blue_k(past(a_{i-1}))| = h_{i-1} + i - 2$. For any honest block $b_j \in past(a_i)$, $|Blue_k(past(b_j))| \le |past(a_i) \cap B| - 1 \le h_i - 1$. So we have $|Blue_k(past(a_{i-1}))| > |Blue_k(past(b_j))|$. a_{i-1} is the highest scoring tip of a_i.

We also have

$$\begin{aligned} &|\text{anti}(b_j) \cap Blue_k(past(a_{i-1}) \cup \{a_{i-1}\})| \\ \le &|\text{anti}(b_j) \cap B| + |\text{anti}(b_j) \cap A| \\ < &k' + k_\Delta. \end{aligned}$$

[1] Assuming synchronous communication between honest nodes simplifies our description and makes our attack stronger. Our attack is of course working for asynchronous network.

So all the block in $B \cap past(a_i)$ will be added to $Blue_k(past(a_i))$, which results in $Blue_k(past(a_i)) \cap B = past(a_i) \cap B$.

If this lemma holds for all the blocks past(b_j), $\forall j' < j, |Blue_k(past(b_{j'}))| \geq j' - 1 - k'$. For any attacker block $a_i \in past(b_j)$, we have $j > h_{i-1+k_\Delta}$ and $b_{h_{i-1+k_\Delta}} \in past(b_j)$. Because

$$\begin{aligned}
&|Blue_k(past(a_i))| \\
=&i + h_i - 1 \\
<&h_{i-1+k_\Delta} - 1 - k' \\
\leq&|Blue_k(past(b_{h_{i-1+k_\Delta}}))| \\
\leq& \max_{b_{j'} \in past(b_j)} |Blue_k(past(b_{j'}))|,
\end{aligned}$$

the highest scoring tip of b_j is in B. We denote it $b_{\bar{j}}$.

For any $a_i \in \text{anti}(G)$,

$$\begin{aligned}
&|\text{anti}(a_i) \cap Blue_k(past(b_{\bar{j}}) \cup \{b_{\bar{j}}\})| \\
\geq&|\text{anti}(a_i) \cap B| - k' \\
\geq&h_{i-1+k_\Delta} - h_i - k' \\
>&k
\end{aligned}$$

No block in $A \cap past(b_j)$ will be add to $Blue_k(past(b_j))$, which means $Blue_k(past(b_j)) \cap A = \emptyset$ □

Theorem 9 *Since the attacker haven't failed, all the honest nodes believe that main chain contains no block in A. However, each time the attacker generates a new block a_i, the attacker can send a_i to all the honest nodes to make every one convince chain be the common main chain.*

Proof. In the previous proof, we show that for each honest block which refers $a_i \in A$ as its parent, this block will regard an honest block as its main chain father. For each local view of a honest node, the largest blue set of honest block is larger than the largest blue set of attacker block.

For any new block a_i, we have

$$\min_{b_j \in past(a_i)} |BLUE_k(past(a_i))| - |BLUE_k(past(b_j))| \geq i - 1.$$

And the main chain of $past(a_i)$ contains all the attacker blocks and no honest block. □

Theorem 10 *Suppose $k_\Delta \geq 6$ and k_Δ is an even integer. The liveness attack never fails with probability larger than zero. It has a lower bound*

$$\left(1 - e^{-cq}\right)^{3k_\Delta - 15} \cdot \prod_{i=3k_\Delta-14}^{\infty} (1 - e^{-q(i-1)}),$$

where $q = \frac{\lambda_a}{\lambda_h}, c = 1.5k_\Delta - 8$.

Proof. It is easy to show that $h_n = \frac{(n-1)(n-2)}{2} + 1$. Now we require attacker to complete the following task.

For a_i with $i \leq 3k_\Delta - 15$, let attacker start to mine a_i when block $b_{(i-1)\cdot c+1}$ is generated. And a_i must be generated before the generation of block $b_{i\cdot c+1}$.

For a_i with $i > h_{2c}$, attacker start to mine a_i when block b_{h_i} is generated and a_i should be mined before the generation of $b_{h_{i+1}}$.

For $i \leq 3k_\Delta - 15$, we have

$$\begin{aligned}
h_i &\leq (i-1) \cdot c + 1 \\
h_{i-1+k_\Delta} &\geq i \cdot c + 1
\end{aligned}$$

For $i \geq 3k_\Delta - 14$, we have

$$\begin{aligned}
h_{3k_\Delta - 14} &= (3k_\Delta - 15) \cdot c + 1 \\
h_{i-1+k_\Delta} &\geq h_{i+1}
\end{aligned}$$

In this task, it can be shown that the mining time intervals of malicious blocks satisfy the requirements in attack strategy and do not overlap. Completing this task implies applying a successful attack. For $i \leq 3k_\Delta - 15$, the attacker fails when mining a_i with probability $1 - (1-q)^c \leq 1 - e^{-cq}$. For $i \geq 3k_\Delta - 14$, the attacker fails when mining a_i with probability $1 - (1-q)^{i-1} \leq 1 - e^{-q(i-1)}$. The probability of completing this task is

$$\left(1 - e^{-cq}\right)^{3k_\Delta - 15} \cdot \prod_{i=3k_\Delta-14}^{\infty} (1 - e^{-q(i-1)}).$$

Let $n = \left\lceil \frac{-\log\left(1-e^{-q}\right)}{q} \right\rceil + 1$, $e^{-qn} < 1 - e^{-q}$, we have

$$\begin{aligned}
&\prod_{i=3k_\Delta-14}^{\infty} (1 - e^{-q(i-1)}) \\
\geq&\left(1 - \sum_{i=n+1}^{\infty} e^{-q(i-1)}\right) \prod_{i=3k_\Delta-14}^{n} (1 - e^{-q(i-1)}) \\
=&\left(1 - \frac{e^{-qn}}{1 - e^{-q}}\right) \prod_{i=3k_\Delta-14}^{n} (1 - e^{-q(i-1)}) \\
>&0
\end{aligned}$$

So the probability is strictly larger than 0. □

As a result, attacker with 15% computation power can maintain this attack infinitely with probability 98.9% when $k_\Delta \geq 40$.