Conflux Protocol: Study, Test, and Implementation

LING Shengchen
Dept. of Computer Science
ID:57730900

HE Jingrao
Dept. of Computer Science
ID:57394554

ZHAI Liuqun
Dept. of Computer Science
ID:57375484

I. Introduction

Blockchain technology has become a popular tool for creating secure, decentralized, and consistent transaction ledgers at an internet scale since the emergence of cryptocurrencies such as Bitcoin and Ethereum. However, despite the expansion of blockchain platforms with the addition of smart contracts, performance limitations persist, especially in terms of throughput and confirmation times. Conflux is a new blockchain system designed to address these challenges and aims to achieve the four desirable properties of security, decentralization, high throughput, and fast confirmation. The innovative consensus protocol and Tree-Graph structure of Conflux eliminate performance bottlenecks and provide robust security against double-spending and liveness attacks. By balancing fast confirmation and consensus progress, Conflux seeks to revolutionize the blockchain landscape, offering a high-performance, secure, and decentralized system suitable for a wide range of applications.

The Nakamoto Consensus, which serves as the fundamental consensus mechanism for blockchain systems like Bitcoin, faces significant limitations. Firstly, it struggles with low transaction throughput and scalability. The fixed block generation rates and limited block sizes of the standard Nakamoto Consensus lead to the processing of a limited number of transactions per second (e.g., around 7 transactions per second for Bitcoin). Secondly, the mechanism experiences latency issues, causing users to wait hours for transaction confirmations. The block generation process takes a specific amount of time (e.g., 10 minutes for Bitcoin) to ensure security, resulting in extended transaction confirmation times. To achieve high confidence in a transaction's irreversibility, users must typically wait for multiple blocks to be added on top of their transaction's block. Additionally, the design of the protocol results in forks and wasted resources since miners occasionally generate blocks simultaneously, leading to temporary forks and discarded blocks. The Proof-of-Work (PoW) mechanism also fosters the growth of mining pools, increasing centralization risks and making the network more vulnerable to manipulation by powerful entities. The PoW mechanism is responsible for high energy consumption, exacerbating environmental concerns and increasing operational costs. Finally, the Nakamoto Consensus remains susceptible to 51% attacks, in which an attacker controlling over 50% of the mining power can disrupt the network, double-spend transactions, and hinder new transaction confirmations.

Conflux presents several advantages over the limitations of Nakamoto Consensus, specifically in terms of transaction throughput, confirmation time, forks, and resource utilization. By allowing multiple participants to contribute to the blockchain simultaneously, Conflux enables faster block generation and higher throughput. It reduces transaction confirmation time by utilizing faster generation rates, enabling blocks to build on top of each other more quickly. Moreover, Conflux processes concurrent blocks without discarding any as forks, improving resource utilization and mitigating issues related to forks in traditional Nakamoto Consensus. Furthermore, Conflux maintains decentralization by enabling multiple participants to contribute to the blockchain concurrently, without creating undesirable hierarchies among protocol participants.

In this report, we investigate Conflux and its consensus protocol, followed by the novel features implemented by Conflux after recent deployment of hard forks. Additionally, we test its basic operations on the Conflux Testnet, which includes sending transactions and deploying smart contracts. Furthermore, we conduct an independent Conflux chain to examine the characteristics of Conflux. Our work can be summarized as follows.

- Our study involves a comprehensive review of papers related to Conflux, focusing on its system design and consensus algorithm. Additionally, we discuss two Conflux Improvement Proposals (CIPs) introduced after the implementation of hard forks.
- We conduct various tests on the Conflux Testnet, including blockchain data retrieval, transaction sending, and smart contract deployment, with corresponding analysis. We also evaluate the performance of Conflux Core and eSpace.
- To further investigate Conflux, we run both a single-node chain and a multiple-node chain, analyzing their configurations and logs in an engineering manner. The multiple-node chain includes one boot node and two new nodes, all of which are full nodes. Our configurations are based on the open-source codes provided by the Conflux team.

The rest of this report is organized as follows. Section II provides an overview of the architecture of Conflux and its consensus protocol. Section III discusses the Conflux Improvement Proposals (CIPs) introduced after the hard forks. In Section IV, we evaluate the basic operations of the Conflux system on the Conflux Testnet. Section V presents the results

and analysis of our independent Conflux chain. Finally, we summarize our findings and draw conclusions in Section VI.

II. System Design of Conflux

In this section, we will provide an in-depth analysis of the architecture of Conflux, its consensus protocol, and the implementation of the consensus algorithm. A thorough understanding of Conflux's system design is crucial for assessing its performance and setting up an independent Conflux chain in the future. Conflux improves on traditional blockchain systems by enabling concurrent block processing without discarding any as forks and delaying transaction total ordering. This approach results in higher transaction throughput and quicker confirmation times, all while maintaining the security and decentralization of the system.

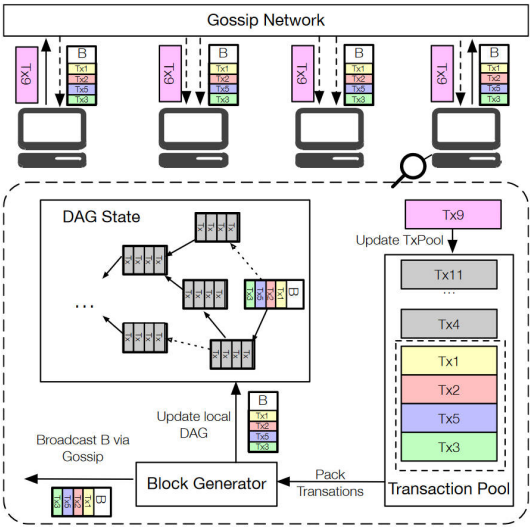

Fig. 1. Architecture of Conflux

A. Conflux Architecture

The Conflux architecture resembles that of Bitcoin, where transactions represent the transfer of coins between payer and payee, identified by public keys. However, Conflux differs by forming a directed acyclic graph (DAG) instead of a linear chain. Each block contains a list of transactions and reference links to previous blocks. Conflux aims to maintain each node's local DAG, enabling all nodes to eventually agree on a total order of blocks and transactions. This allows for the support of smart contracts, which execute Turing complete programs to update account states. The diagram of Conflux's architecture is depicted in Fig. 1. A Conflux system comprises three main components, including the Gossip Network, Pending Transaction Pool, and Block Generator.

- **Gossip Network.** Conflux nodes are linked through a gossip network responsible for distributing transactions and blocks. This network is constructed using a modified version of the implementation used in Bitcoin Core.
- **Pending Transaction Pool.** Each node maintains a pending transaction pool that stores transactions not yet included in any block. When a new transaction is received or a new block is discovered, the node updates its pending transaction pool accordingly. The consensus protocol manages any duplicate or conflicting transactions that may arise due to network delays or malicious nodes.
- **Block Generator.** Conflux nodes employ a block generator to create new blocks for packing pending transactions. The system uses a proof-of-work (PoW) mechanism similar to Bitcoin to ensure a stable network block generation rate by dynamically adjusting the PoW problem difficulty. Additionally, Conflux is compatible with alternative mechanisms, such as proof-of-stake (PoS), that also maintain a consistent block generation rate.

B. Consensus Protocol

The Conflux consensus protocol functions on each individual node's local DAG state, with the objective of attaining consensus on the total order of block generation among all nodes. Fig. 2 depicts a local DAG state of a Conflux node. Aside from the genesis block, all other blocks have only one outgoing parent edge and can have multiple reference edges, indicating the chronological order of the blocks. Conflux divides all blocks within a DAG into epochs, with each block on the main chain being responsible for one epoch. The protocol first sorts the epochs and then sorts the blocks within each epoch. Any unordered blocks are sorted based on their hash values. To achieve consensus on the total order of newly generated blocks, the following three issues must be addressed.

(1) How to select the pivot chain?

Unlike Nakamoto Consensus, which uses the longest chain rule to maintain consensus and resolve conflicts between nodes, Conflux utilizes the GHOST (Greedy Heaviest Observed Subtree) rule to select the Pivot Chain. The GHOST rule operates on the parental tree, which comprises all parent edges in a DAG with the genesis block as the root. The process of selecting the pivot chain commences at the genesis block and follows these steps: During each phase, the subtree sizes of all child blocks in the parental tree are first computed, and then the child block with the largest subtree size is selected. The path extending from the genesis block to the leaf block, identified using the GHOST rule, constitutes the pivot chain.

(2) How to generate new blocks?

When generating a new block in Conflux, a node first computes the pivot chain within its local DAG state, designating the final block of the chain as the parent of the new block. This operation adds weight to the chain, incentivizing other nodes to select the same pivot chain in the future. The node subsequently identifies all DAG tip blocks without incoming edges and creates reference edges linking the new block to each of these tip blocks.

(3) How to solve the double-spending transactions?

In Conflux, conflicting transactions are resolved by initially sorting them based on their block order. Transactions within the same block are then ordered by their position. Conflux identifies conflicts during this process, discarding the later conflicting transaction. If a transaction is present in multiple

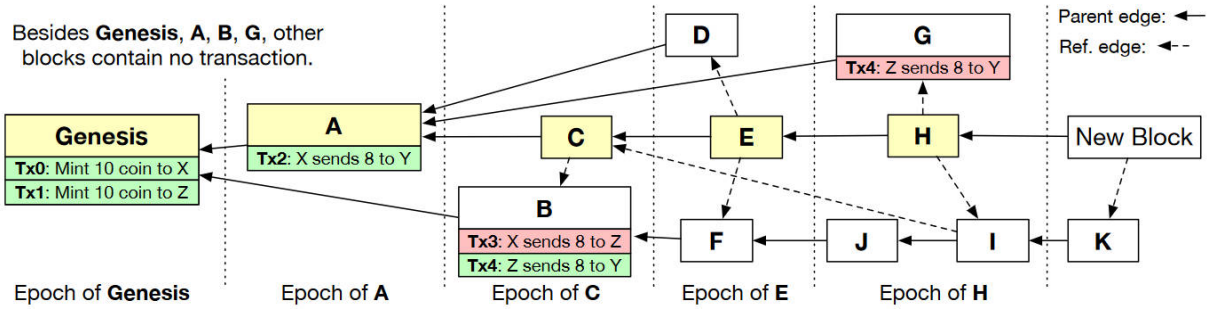

Fig. 2. An example to demonstrate the consensus algorithm in Conflux.

blocks, only the first instance is retained, and the others are eliminated.

C. Implementation of Conflux Consensus

In this section, we provide a detailed analysis of the Conflux consensus mechanism and investigate how consensus is reached among multiple nodes in the system. The local state of a node in Conflux can be represented as a Directed Acyclic Graph $S =< B, g >$, where B is the node's block collection and g represents the genesis block of Conflux. A block $b \in B$ is connected to its parent block via a parent edge and to other blocks through reference edges that indicate the generation order. To achieve consensus, Conflux first selects a Pivot Chain and then sorts the blocks in the DAG to confirm the order of block generation. Then, Conflux determines the transaction order while handling conflicting transactions. This section briefly describes the implementation of the two most important algorithms in the consensus protocol, namely, the *Pivot(B, b)* function responsible for selecting the Pivot Chain and the *ConfluxOrder()* function responsible for sorting the blocks.

(1) *Pivot(B, b)*

Input : A set of blocks B and a starting block b.
Output : The pivot block for the subtree of b.

1 **if** $\mathsf{Child}(B,b) = \emptyset$ **then**
2 **return** b
3 **else**
4 $w \leftarrow \max\{\mathsf{SubTW}(B,i) \mid i \in \mathsf{Child}(B,b)\}$
5 $a \leftarrow \underset{i \in \mathsf{Child}(B,b)}{\arg\min} \{i.\mathsf{hash} \mid \mathsf{SubTW}(B,i) = w\}$
6 **return** $\mathsf{Pivot}(B,a)$

Fig. 3. The definition of $Pivot(B,b)$

The *Pivot(B, b)* function is a recursive algorithm that takes as input a set of blocks B and the genesis block g and outputs the leaf block of the chosen pivot chain. To select the pivot chain, the algorithm iteratively selects the child block with the largest subtree weight, using the smallest unique hash id to break ties. The process continues until a leaf block is reached.

During the Pivot Chain selection process, it is essential to calculate the weight of a block. The Conflux consensus mechanism employs the *AdaptiveWeight(b)* function to compute the weight of a block. This function first checks if a block requires adaptive weight based on its past block set. If not, the weight is set to one. If adaptive weight is necessary, the function examines the subtree graph, ensuring progress by making a subtree dominant after a specific time. This approach incentivizes honest participants to generate blocks with adaptive weights and adopts a conservative strategy similar to the structured GHOST algorithm to prevent attacks.

(2) *ConfluxOrder()*

Input : The local state $S = \langle B, g \rangle$ and a new discovered block b

1 **if** $b.\mathsf{pow\_quality} \geq D$ **then**
2 Wait until $\mathsf{Past}(b) \subseteq B$
3 **if** $\mathsf{Pivot}(\mathsf{Past}(b), g) = b.\mathsf{parent}$ **then**
4 $b.\mathsf{weight} \leftarrow \mathsf{AdaptiveWeight}(b)$
5 $S \leftarrow \langle B \cup \{b\}, g \rangle$

Figure 7: The block validation procedure.

Input : A block b
Output : An ordered list of all blocks in $\mathsf{Past}(b) \cup \{b\}$

1 **if** $b.\mathsf{parent} = \mathsf{Nil}$ **then**
2 **return** $\emptyset$
3 $L \leftarrow \mathsf{ConfluxOrder}(b.\mathsf{parent})$
4 $B_\Delta \leftarrow (\mathsf{Past}(b) - \mathsf{Past}(b.\mathsf{parent}) - \{b.\mathsf{parent}\}) \cup \{b\}$
5 **while** $B_\Delta \neq \emptyset$ **do**
6 $B'_\Delta \leftarrow \{x \mid |x.\mathsf{pred\_blocks} \cap B_\Delta| = 0\}$
7 Sort all blocks in B'_Δ in order as $a_1, a_2, \ldots, a_k$
8 such that $\forall 1 \leq i < j \leq k, a_i.\mathsf{hash} < a_j.\mathsf{hash}$
9 $L \leftarrow L \circ a_1 \circ a_2 \circ \ldots \circ a_k$
10 $B_\Delta \leftarrow B_\Delta - B'_\Delta$
11 **return** L

Fig. 4. The definition of $ConfluxOrder()$

The *ConfluxOrder()* function is utilized to determine the order of all blocks that appear in or before a specific block epoch. The function operates recursively in reverse, beginning with finding the parent block of block b and assigning the order within the parent block's epoch to L. Subsequently, the order of all blocks within the epoch of b is calculated.

To illustrate, consider Fig. 2 as an example, where we aim to sort multiple blocks within the epoch containing block H on the Pivot Chain. Within H's epoch, blocks G and J have no outgoing edges. Sorting them by their hash values, G and J

are added to L. After removing these two blocks, we sort the remaining blocks H and I. Since I has no outgoing edges after removal, and H points to I, I is placed before H, resulting in the order (G, J, I, H). This same method is applied to sort blocks in other epochs.

D. Analysis of Conflux

Conflux is a blockchain system that prioritizes scalability and decentralization, with a focus on achieving high throughput and fast transaction confirmation. To accomplish this goal, Conflux employs an innovative consensus protocol that enables concurrent block processing while preserving all blocks and incorporates an adaptive weight mechanism that dynamically assigns weights to blocks based on their topologies within the Tree-Graph structure. This adaptive weight mechanism is effective at identifying and mitigating liveness attacks, as it can switch between optimistic and conservative strategies as needed.

In summary, Conflux has three innovative features.

- **Novel consensus protocol.**
  Conflux improves upon the traditional Nakamoto Consensus by proposing a new protocol based on a DAG structure that doesn't discard forked blocks.
- **Optimized transaction sorting rules.**
  Conflux first sorts blocks by epoch, resolves conflicting or double-spending transactions, and then sorts transactions within the blocks.
- **High throughput and short confirmation time.**
  Conflux has achieved high transaction throughput, processing thousands of transactions per second. This speed is over ten times that of Bitcoin, while maintaining the same level of security as traditional Nakamoto Consensus.

III. New Features after Hard Fork

Apart from the system design proposed in the papers [1] [2], Conflux has successfully implemented two hard forks to its Mainnet, launching several more novel features. On February 2022, the Hydra hard fork implemented 8 CIPs including CIP-43 and CIP-90 [3]. In this section, we mainly focus on these two CIPs and their corresponding features.

A. CIP-43: Introduce PoS Finality [4]

1) Motivation: In the early stage of any PoW blockchain, when the total amount of computational power of the entire network is relatively insufficient, it's easy for attackers to launch 51% attacks with a lower cost and steal the on-chain assets by double-spending attacks. In the years of 2020 and 2021, Ethereum Classic [5], Grin [6], and Verge [7] all had problems with 51% attacks. Therefore, Conflux has to consider the reality of the 51% attack risk for quite a long time.

Theoretically, the blockchains using PoW can never completely eliminate the risk of 51% attacks. However, in practice, whether an attacker launches a 51% attack mainly depends on the balance between benefits and costs.

In terms of cutting attacker's benefits, due to the characteristic of temper-proof of blockchains, a successful 51% attacker will not lose on-chain revenue (tokens), yet the corresponding value may still drop due to two reasons. One is the loss caused by the sharp drop in token value after the attack; the other is the community reaching a consensus to roll back the state through a hard fork to eliminate the attacker's income. However, due to the development of cross-chain technology, it is difficult to eliminate the impact of double-spending attacks through simple hard forks, which is quite different from the situation when ETH and ETC forked.

In terms of increasing attacker's costs, there are three options available: 1) Avoid all scaled mining pools, specifically, abandon graphics card mining and switch to dedicated ASIC mining. 2) Enable the computational power of Conflux to approach or even surpass that of Ethereum; in other words, increase Conflux's total mining rewards to a level comparable to that of Ethereum. 3) Introduce a Proof-of-Stake (PoS) mechanism to check and balance 51% attack, which makes it impossible for an attacker to achieve a double-spending attack based solely on the advantage of computational power, unless the attacker invests enough on-chain assets to affect the PoS decision. Of the three options above, only the last option is actually feasible.

As a result, it is natural to add PoS finality decision to Conflux's existing PoW consensus mechanism, to increase the confidence of high-value transactions on Conflux, and resist potential 51% attacks with high possibility.

2) Specification: Conflux introduced PoS finality to its Mainnet in a hard fork named "Hydra" on February 2022.

PoS Consensus. Instead of thoroughly switching the consensus mechanism to PoS, Conflux introduces a stand-alone PoS chain to reach a consensus on the previous pivot blocks on PoW chain. Once a pivot block is confirmed by PoS chain, all PoW miners should follow it. This means that even if a 51% attacker attempts to reverse the block, other PoW miners will not approve and follow it, see Fig.5 for details.

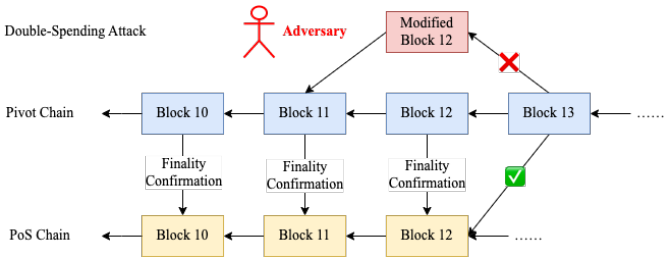

Fig. 5. PoS enhances security

PoS finality requires CFX (the native token of Conflux) holders to stake CFX to a specific smart contract to claim corresponding votes on the PoS chain, and the PoS chain uses a verifiable random function (VRF) to elect a 300-member committee, which is temporary and rotates regularly. The committee consists of 6 sessions of members, each containing 50 seats. Every 1 hour, the longest-serving session retires

and a new session takes over. During the election period, the PoS accounts submit the results of VRF, and hash values are calculated according to the results. The 50 votes with the smallest hash values are selected.

For stability consideration, PoS stakeholders are allowed to un-stake from the smart contracts only when 13 days after staking, and finally withdraw the assets another 1 day later. This prevents the stakeholders, especially the committee members, from frequently staking and withdrawing.

Financial Incentives. Conflux designed a 4% inflation annually in the previous tokenomics. After the hard fork, PoS staking rewards add a coefficient on top of the existing 4%. Let $x = \frac{totalCirculationOfCFX}{totalStakingOfCFX}$, and the coefficient is set to $\sqrt{x}$. For instance, when the staked amount is $\frac{1}{4}$ of the circulating supply, the reward rate is 8%; when the staked amount is $\frac{1}{9}$, the reward rate is 12%; so and so forth. If the proportion of stakeholders is relatively low, those who choose to stake will get more benefits. Such a design aims to incentivize CFX holders to participate in staking and PoS voting, thereby fortifying the network's security and stability.

Relationship between PoW and PoS. In order to reduce the complexity and potential risks of adding PoS finality, after the hard fork, PoW miners are still completely responsible for packaging transactions, and block ordering still follows the Tree-Graph ordering rules. The newly-added PoS mechanism only affects the finality of blocks generated by PoW miners.

3) Security Considerations: In a theoretical stage, after the hard fork, the PoW + PoS consensus protocol can provide security in different scenarios.

51% Attack. If the attacker controls more than 51% computational power and the PoS chain works well, the pivot blocks that have been decided by the PoS chain are still safe. However, the PoW chain may lose liveness because the PoS nodes cannot confirm any new blocks.

34% Staking Attack. If the attacker controls more than 34% PoS staking tokens, the PoS chain may lose liveness, but it will not affect the consensus of PoW chain, the blockchain can still be considered safe.

67% Staking Attack. If the attacker controls more than 67% committee members, it can generate conflict PoS blocks. In this case, the PoW chain will diverge, even if the attacker disappears afterward.

B. CIP-90: Introduce EVM-Compatibility [8]

1) Motivation: Ethereum is currently the largest smart contract platform. The Ethereum Virtual Machine (EVM) is the core of Ethereum, every node of the network runs the EVM and executes the same instructions on the blockchain [9]. In EVM, smart contracts use a high-level Turing-complete language called Solidity, which is logically similar to JavaScript. Smart contracts are compiled into EVM bytecodes and deployed on EVM.

However, Conflux uses a Conflux Virtual Machine (CVM), similar to EVM but exist some differences [10]. For instance, Conflux adopts a Collateral for Storage (CFS) mechanism. When executing the smart contract, if non-zero data is written to the storage via SSTORE (0x55) command, storage collateral is required to be paid, pricing 1 CFX per KB. When this data is cleared, the collateral will be refunded. Moreover, Conflux has a different transaction format and a different rule for generating addresses from public keys. These differences make it hard to migrate EVM-compatible decentralized applications (dApps) to Conflux. For other layer-2 blockchains, e.g., Polygon [11], Arbitrum [12], and zkSync [13], most of which remain EVM-compatible.

The previous CIP-72 [14] and CIP-80 [15] tried to solve the problems. CIP-72 planned to enable Conflux to accept the signatures signed by Ethereum wallets, e.g., Metamask, while CIP-80 planned to enable Conflux to produce the same address format as Ethereum with the same private key. However, these changes will affect the current dApps, all current applications would have to modify their source codes to fit in the changes, which is less realistic. Therefore, Conflux team saw a necessity to introduce EVM compatibility for the purpose of ecosystem development and convenience of developers, which is meanwhile reasonable and acceptable.

2) Specification: Conflux also introduced a new EVM-compatible space in the same hard fork as Section A. The newly-introduced space is called Conflux eSpace (eSpace for short), and the original space is called Conflux Core (Core for short).

Addresses. The two addresses generated in the two spaces share the same private key, however, are isolated logically with independent balances and statuses. The Conflux Core still remains base32 format addresses, e.g., "cfx:aaj0r6he9uz5dkb7hzhmpybnsxdzxw11mpm64f9vm9", while the Conflux eSpace uses hex40 addresses, e.g., "0x5d89e5eCcd773Cd72d0858e60f4E0C64aB84e594".

Cross-chain bridge. The two addresses can interact with each other via a cross-space bridge with atomicity and layer-1 security. This is implemented through an internal contract called "CrossSpaceCall" deployed at the address "cfx:aaejuaaaaaaaaaaaaaaaaaaaaaaaaaaa2sn102vjv" (0x0888000000000000000000000000000000000006) with a series of preset interfaces [16].

Throughput. According to CIP-90, only the blocks with an epoch number of multiples of 5 can package eSpace transactions, and the total gas limit of these transactions cannot exceed half of the block gas limit. Therefore, the throughput of Conflux eSpace is around 10% of which in the previous Conflux protocol, theoretically. See Fig.6 for details.

Compatibility. eSpace is fully compatible with Ethereum at the interface level, including RPC and EVM. The basic development tools of Ethereum (e.g., Metamask, truffle, hardhat, Remix, ethers.js, web3.js) can be directly used in eSpace. However, there still exist some slight differences [17] but most will not seriously affect migration.

3) Ecosystem Evaluation: The ecosystem development has seen a range of prosperity since the hard fork [18]. By the time this report is drafted, the basic decentralized finance (DeFi) infrastructures have been implemented, e.g., decentralized exchange Swappi, lending protocol Goledo, stablecoin protocol

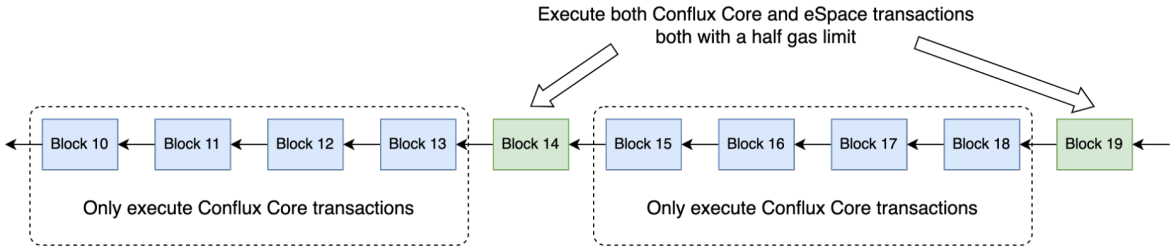

Fig. 6. Decreased throughput in eSpace due to gas limit

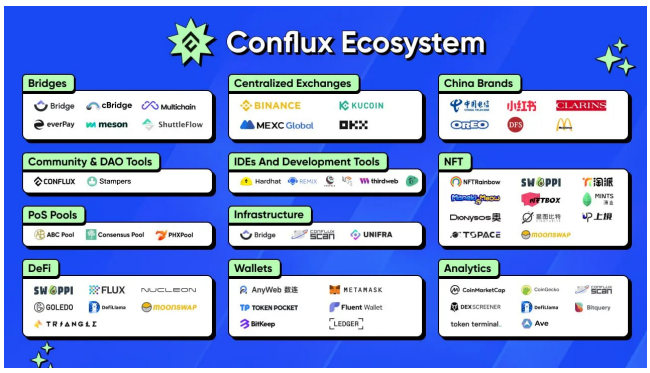

Fig. 7. Conflux ecosystem dApps [18]

TriAngleDAO, and Conflux has connected with other EVM-compatible chains via cross-chain bridges like Multichain, see Fig.7 for details. On February 2023, Conflux also announced its cooperation with China Telecom on blockchain-based SIM cards in Hong Kong [19].

IV. Test Operations on Testnet

In this section, we test some basic operations to interact with Conflux Testnet in order to comprehend Conflux better, including retrieving blockchain data, sending transactions, and deploying smart contracts, with corresponding analysis. Specifically, considering the new features mentioned above, we conduct the tests both in Core and eSpace. Finally, we do some performance testing on Conflux.

One thing that requires attention is that Conflux Mainnet has been running stably since October 2020, making it challenging to test past issues with a high degree of certainty. As a result, this section represents an opportunity for us to gain a deeper understanding of blockchain technology and advance our skills in developing on a blockchain platform. We suggest developers, especially beginners, go through our process to establish a scope of development, and go to Conflux Documentations [20] for details.

A. Setting up

To obtain the pertinent data, we mostly use the JSON-RPC protocol and the js-conflux-sdk. The necessary environment should first be well configured before we start testing. Moreover, we use ChainIDE and two plugin wallets, i.e., Fluent Wallet and Metamask.

1) JSON-RPC: Remote Procedure Call (RPC) is a protocol that enables communications between client and server applications over a network. A simple data interchange format linked to JSON is called JSON-RPC. The JSON data that the interface returns can be obtained using JSON-RPC. We use the JSON-RPC offered by the Conflux team to interact with the blockchain without running a node.

2) JavaScript runtime environment: The JavaScript runtime environment, of which node and npm are the most crucial components, is first installed in order to utilize js-conflux-sdk. A fairly developed set of package management tools is npm. We can rapidly download the js-conflux-sdk by using npm. We use Node, an open-source, cross-platform js runtime environment, to execute js scripts.

3) js-conflux-sdk: Software Development Kit (SDK) provides a set of tools and resources that help streamline the development process by providing pre-built functionality that can be easily integrated into an application. Conflux team offers several SDKs including js-conflux-sdk, go-conflux-sdk, java-conflux-sdk, and python-conflux-sdk. These SDKs provide developers with a range of options to choose from based on their preferred programming language and development environment. Overall, Conflux SDKs make it easier for developers to build blockchain-based applications on Conflux. In this section, we use js-conflux-sdk as an example to implement the tests, which we are most familiar with.

4) ChainIDE: For smart contract testing, we edit and deploy smart contract codes using the environment offered by ChainIDE. ChainIDE is a cloud-based integrated development environment (IDE) designed for blockchain developers.

5) Plugin wallets: We use Fluent Wallet, the Conflux plugin wallet, to manage our blockchain accounts and connect with ChainIDE. Fluent Wallet supports both Conflux Core and eSpace, however, due to the limitation of ChainIDE, we use Fluent Wallet to interact with Conflux Core while using Metamask to interact with eSpace.

B. Transaction Testing

Our tests are implemented on Windows 10 platform via CityU WiFi, and the target network is Conflux Testnet v2.2.3.

1) Retrieving blockchain data: js-conflux-sdk offers us a highly comprehensive API interface that enables us to access various blockchain data and carry out additional actions on the chain. We can acquire nonce, balance, blocks, and other

information through these interfaces in addition to the ones that are most frequently used for transmitting transactions, and we may analyze the data that was returned by the interface. In this part, we use the API interface to gather some blockchain-related data in order to get ready to submit transactions later on. Fig.8,9,10 depict the outputs of obtaining the nonce, balance, and blocks according to epochNumber. Fig.11 shows the basic information of the blockchain.

```
$ node main.js
{
  data: {
    jsonrpc: '2.0',
    id: '187709ce521b696b8c6c524b',
    method: 'cfx_getNextNonce',
    params: [ 'cfxtest:aanycwkmgw1gedjk9dmrp6k3h7f5m2zmuyd5wtaan0' ]
  },
  result: '0xc92',
  duration: 1546
}
nonce: 3218
```

Fig. 8. Get Next Nonce

```
{
  data: {
    jsonrpc: '2.0',
    id: '187709e0d974f03c30cb678',
    method: 'cfx_getBalance',
    params: [ 'cfxtest:aanycwkmgw1gedjk9dmrp6k3h7f5m2zmuyd5wtaan0' ]
  },
  result: '0xd51a279df7b0b3889800',
  duration: 1594
}
Balance: 1006346530907228000000000
```

Fig. 9. Get Balance

```
{
  data: {
    jsonrpc: '2.0',
    id: '18770a334981d5f3844e1e68',
    method: 'cfx_getBlocksByEpoch',
    params: [ '0x710fb19' ]
  },
  result: [
    '0xed11daa4006ee70acdb40873ab5e2146c401d981de5069bdbe368b55f1ce6435',
    '0x880a8a815b5c2c9f1ba1d40ae82722c87b33b5cac0d7e99a6789373a4e5390da'
  ],
  duration: 1565
}
```

Fig. 10. Get Blocks by Epoch Number

```
{
  data: {
    jsonrpc: '2.0',
    id: '18770a0a2e6387a1ce44bfd7',
    method: 'cfx_getStatus',
    params: []
  },
  result: {
    bestHash: '0x487bf2ecae853b5c1ad5588b199049eef06f77374e7b5678482a44ddb130085
1',
    chainId: '0x1',
    ethereumSpaceChainId: '0x47',
    networkId: '0x1',
    epochNumber: '0x7110c78',
    blockNumber: '0x91697f7',
    pendingTxNumber: '0x224',
    latestCheckpoint: '0x70f8e60',
    latestConfirmed: '0x7110c31',
    latestState: '0x7110c74',
    latestFinalized: '0x7110a74'
  },
  duration: 861
}
```

Fig. 11. Initialization Status

2) Sending a transaction: Since sending transactions is a fundamental operation in blockchain technology, we begin by attempting to send a transaction. The critical component of this process is depicted in Fig.12. The information returned by the transaction is shown in Fig.13.

```
let txParams = {
  from: account, // from account instance and will by sign by acco
  // nonce: 5202,
  // gasPrice: 100000000000,
  // gasPrice: 1000000000000,
  // gas: 100,
  to: receiver, // accept address string or account instance
  value: Drip.fromCFX(1), // use the conversion utility function
  // storageLimit
  // epochHeight
  // data
};
async function main() {

  const txHash = await conflux.cfx.sendTransaction(txParams);

  console.log('txHash: ' + txHash);
}
```

Fig. 12. Part of Code

```
{
  data: {
    jsonrpc: '2.0',
    id: '18769247d9926bc53ab61525',
    method: 'txpool_nextNonce',
    params: [ 'cfxtest:aanycwkmgw1gedjk9dmrp6k3h7f5m2zmuyd5wtaan0' ]
  },
  result: '0xc70',
  duration: 272
}
{
  data: {
    jsonrpc: '2.0',
    id: '18769247ebc8f36184cd02f8',
    method: 'cfx_epochNumber',
    params: []
  },
  result: '0x70e2ef2',
  duration: 217
}
{
  data: {
    jsonrpc: '2.0',
    id: '18769247f9cf57122b9f67b7',
    method: 'cfx_gasPrice',
    params: []
  },
  result: '0x3b9aca00',
  duration: 226
}
{
  data: {
    jsonrpc: '2.0',
    id: '187692480c4e7fab33b7b69a',
    method: 'cfx_sendRawTransaction',
    params: [
      '0xf875f1820c70843b9aca00852508941166f0e4fc2bb1a43d3d4ea6502b74c759caf753880de0b6b3a
65f639844b332fb173ecea89c3540da3ffe2183ed58274f6ae74433033'
    ]
  },
  result: '0x3e4de9eb4ff73555ceae490f9856c378a1b4958741418556a74f06e17a68ed20',
  duration: 212
}
time for this transaction: 1033ms
txHash: 0x3e4de9eb4ff73555ceae490f9856c378a1b4958741418556a74f06e17a68ed20
```

Fig. 13. Transaction Message

3) Analysis of transactions: In Fig.13, there are 4 functions executed in the process of sending the transaction: *txpool-nextNonce*, *cfx-epochNumber*, *cfx-gasPrice*, *cfx-sendRawTransaction*. We will briefly introduce them below.

- *txpool-nextNonce* is to get the next nonce. That is, get the counter number that is required to be used for the next transaction.
- *cfx-epochNumber* shows the current epoch height.
- *cfx-gasPrice* shows the gas price of the transaction.
- *cfx-sendRawTransaction* is the method to actually send the transaction, in which the hash value is returned via result, while params return the entity of this transaction. It includes the following contents in Fig.14, which are processed by serialization methods to form a string and then wrapped in hexadecimal format and returned.

4) Transaction Stages: For a complete Conflux transaction, there are several processes to go through. Fig.16 shows the

Fig. 14. Returned Message

specific flow chart of the whole transaction process.

- Ascertain address and secret key (from). Conflux can derive the sender's address by obtaining the private key during the transaction.
- Form the transaction data, compile the transaction data, put it all together, sign and encrypt it, and then send it. The structure of a transaction is shown in Fig.15, where the terms from, to, and value are crucial.

  - to: the recipient of the transaction
  - nonce: the sequence number of the transaction
  - value: the transaction amount, valued in Drip
  - data: transaction data
  - chainId: the chain ID of the transaction execution
  - epochHeight: the height that the transaction execution targets
  - gas: maximum gas amount
  - gasPrice: the gas price
  - storageLimit: storage staking limit

Fig. 15. The Format of a Transaction

- Send the RawTransaction to full nodes using the RPC method cfx-sendRawTransaction. A transaction failure will be returned if the created transaction is flawed, but if it passes the checksum, it will be added to the transaction pool and its hash value will be returned. The transaction is currently in the processing stage but has not yet been successfully completed.
- Package the transaction. The miners will package transactions that satisfy the requirements and verifications, and the gas price is a key factor in the packaging order.
- Delayed gratification. Conflux uses a block execution delay mechanism, that is, when a block is packaged, it is not immediately performed but rather after a wait of 5 epochs, the purpose of which is to complete all transactions in the block.
- Awaiting the confirmation of roughly 50 Epochs. The mere fact that a transaction has been completed does not guarantee that its status won't change in the future. Due to the linked nature of the blockchain, when new blocks are added or removed, the main chain may fork or shift, potentially restoring certain transactions. A fixed number of additional blocks must often be created after the transaction blocks have been packed in order to finally confirm the transaction.
- Awaiting the PoS chain to mention it. The PoW blocks are voted as finalized at this point by the committee in the independent PoS chain.

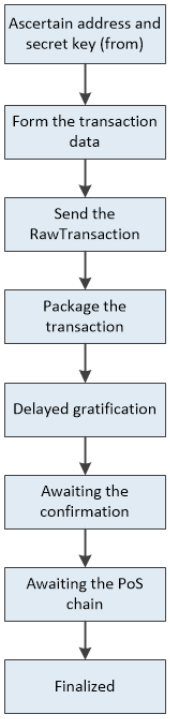

Fig. 16. The Whole Stage of Sending a Transaction

C. Smart Contract Testing

Apart from basic transactions, Conflux also acts as a smart contract platform. Therefore, we try to write and deploy some smart contracts following several token standards, including fungible token standards CRC-20 and ERC-20, and non-fungible token standards CRC-721 and ERC-1155. We choose these because cryptocurrencies and NFTs are widely used in dApps.

1) CRC-20 token: The CRC-20 standard is derived from ERC-20, which is specially designed for Conflux. Following CRC-20, we deploy a contract with its contract address "cfxtest:aca979pkc3rnjgsm27kvvsa8u7c7b9b0jpfmyt5hhb", and four internal transactions are generated in this process. See Fig.17,18.

2) CRC-721 token: We also deploy a contract following CRC-721. For this contract, we set name=CS6290 and symbol=CS, which are the parameters of the constructor, with its contract address "cfxtest:acddke192z3wyrcwcahbgjkv6ybm507h5uhptm5jzt". See Fig.19.

3) ERC-20 token: In order to verify the EVM compatibility of eSpace, we deploy a contract following ERC-20, which is a standard in Ethereum, with the contract address "0xbf7c2950c19ea1641023198e3d23c1147daa14f8".

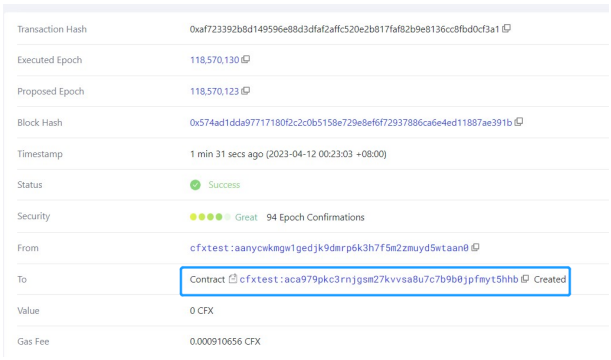

Fig. 17. The Record of CRC-20 Contract

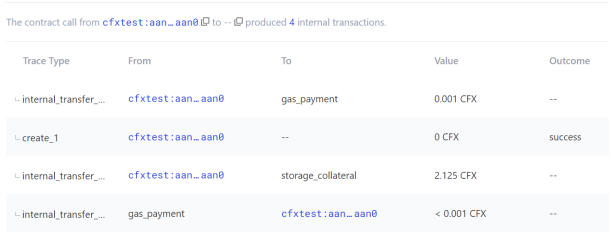

Fig. 18. The Transactions of CRC-20 Contract

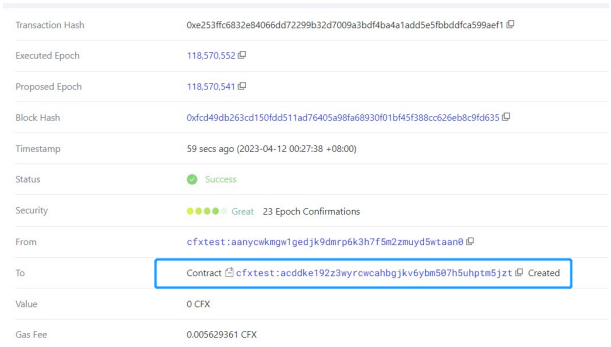

Fig. 19. The Record of CRC-721 Contract

See Fig.20. Note that the address here also follows the format of Ethereum.

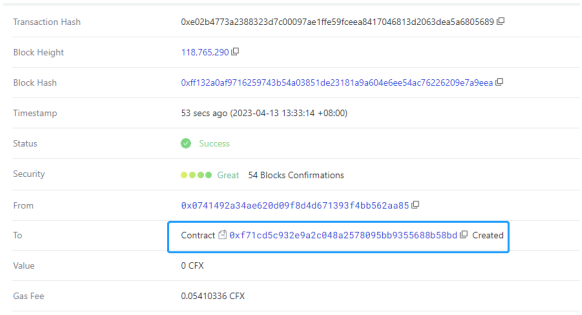

Fig. 20. The Record of ERC-20 Contract

4) ERC-1155 tokens: Similar to the above, a contract following ERC-1155 is also deployed, with the contract address "0xf71cd5c932e9a2c048a2578095bb9355688b58bd". See Fig.21.

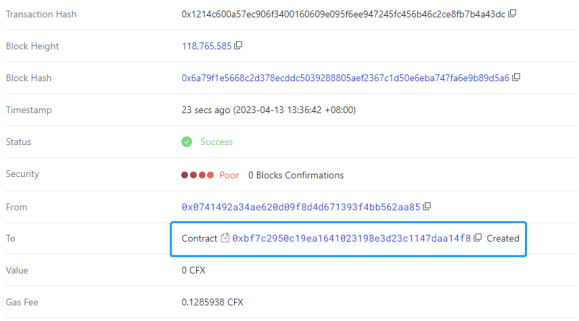

Fig. 21. The Record of ERC-1155 Contract

D. Performance Testing

We further evaluate the performance of Conflux by sending multiple transactions in both Conflux Core and eSpace, in the unit of transactions per second (TPS). To the best of our knowledge, there is no experimental result of the performance of eSpace after the hard fork.

1) Conflux Core Performance: For Conflux Core testing, we try to send 500 transactions simultaneously by JS scripts. This is achieved by using a loop and modifying the value of the nonce cyclically. To determine the time the transactions took to execute, we collect all the *blockHash* that are returned and look up their numbers.

The transaction sender specifies the gas fee by setting the gas and gasPrice parameters of the transaction. The gas field is used to set the maximum amount of gas that may be paid for the transaction, 21000 by default; while the gasPrice field is to set the price of a single gas in the unit of drip, which is the smallest unit of CFX token, and 1 drip equals 10^{-18} CFX. During transaction execution, the upfront gas fee is determined as follows: gas fee = gas * gasPrice. According to the characteristic of blockchain, the packaging speed of a transaction should be affected by the value of the gas fee. Therefore, we gradually change the value of the gasPrice.

- First, we use the default value of 1G drip (1G=10^9). The transactions are executed in 278 blocks with the equivalent TPS of 3.59. Part of the detailed results can be seen in Fig.22.
- Then we let gasPrice = 100G drip. In this case, the transactions are executed in 274 blocks, with the equivalent TPS of 3.65.
- Then we expand gasPrice again, let gasPrice = 1000G drip. In this case, the transactions are executed in 247 blocks, with the equivalent TPS of 4.01.

As can be seen, the increase in the gas fee can indeed affect the efficiency with which transactions are executed, and priority is always given to transactions with higher gas fees.

2) Conflux eSpace Performance: For Conflux eSpace testing, as its theoretical performance is only about 10% of that in the previous Conflux, we try to send 100 transactions

```
{
  data: {
    jsonrpc: '2.0',
    id: '18766f395ba549c58f76f329',
    method: 'cfx_sendRawTransaction',
    params: [
      '0xf875f18204f7843b9aca00825208941166f0e4fc2bb1a43d3d4ea(
    ]
  },
  result: '0x8d06286b4b07b5d0269a7b85c3b0ed06c5507b8d367cdeb
  duration: 266
}
time for this transaction: 296ms
txHash: 0x8d06286b4b07b5d0269a7b85c3b0ed06c5507b8d367cdeb
nonce: 1272
```

Fig. 22. The Part of Result of 500 Transactions

simultaneously. This is achieved by deploying a simple contract and calling it using js-conflux-sdk. The contract address is "0x9624bd1e6547ce5d53d09d336f87fd19f67fb18d" and has been verified on ConfluxScan [21]. However, the execution results are much slower than expected. For 100 transactions, the average execution time is 7457ms, with the equivalent TPS of 0.13.

3) Analysis: The results of our tests are far lower than the experimental results in the papers [2] [1], from our analysis, the reasons for which can be as follows.

- We use js scripts to interact with Conflux, however, JavaScript is single-threaded, and its performance is not good enough when processing multiple transactions, which may affect the time of transaction sending, thereby affecting performance. For improvements, we suggest using GO and go-conflux-sdk.
- We use RPC services to interact with Conflux, however, there may exist some query limitations when sending large amounts of transactions simultaneously.
- The experimental results in the paper [2] are achieved by turning off signature verification and transaction execution to ensure enough computation resources, with a mining speed of 4 blocks per second. For our testing condition, our transactions are executed with the full process and the actual mining speed is 2 blocks per second.
- Our contract codes and scripts logics can also be improved from the perspective of performance and efficiency.

V. Run an Independent Chain

To deeply understand how a blockchain node works as well as the Tree-Graph consensus mechanism, we run an independent Conflux chain and summarize our findings. Our blockchain is a private chain and does not connect with any existing blockchain. As itself, the chain acts as an infrastructure of Web3 and a direct implementation of Conflux Protocol.

We are fully aware that our work may not be perfect and comprehensive, however, we do have a deeper understanding of the internal mechanisms in the engineering manner.

A. Run a Single Node Chain

We run the single-node chain based on the open-sourced codes of Conflux v2.2.0 by modifying configurations. We set the "bootnodes" parameter to empty thus the node will not seek any boot node. We also modify the "dev_block_interval_ms" parameter to customize the blockchain generation interval. This process can be seen as preparation work for the next section.

B. Run a Multiple Node Chain

We successfully run the multiple-node chain on Windows 10 platform on three laptops on the same LAN, and gain a few findings during the process. Still, we run the multiple-node chain based on the open-sourced codes of Conflux v2.2.0 by modifying configurations.

Step 1: Configure a boot node. As the first node in the new network, we comment the "bootnodes" parameter. We set the "dev_allow_phase_change_without_peer" to "true" to enable the node to start mining when no peer is connected. We add the IP address of the machine to "public_address" parameter and open "public_tcp_port=32323" to enable other nodes to connect with the boot node, specifically for our boot node is "192.168.43.142:32323". We also open "jsonrpc_http_port=12537" to enable RPC interactions, specifically for our boot node is "192.168.43.142:12537". Further, we set the "min_phase_change_normal_peer_count" parameter to "1" to enable the network to enter the normal phase when there's only one boot node. For this network, we set the "chain_id" as "6290" and "evm_chain_id" as "6291".

Step 2: Run the boot node. Up to now, we can run the boot node for the first time. We open PowerShell under the node directory, and enter the command "conflux.exe –config bootnode.toml" to run with our own configurations.

Step 3: Connect the wallet. After running the boot node, we can see the processes in the command line window. However, the node is not mining as we have not set the mining configurations. We enter Fluent Wallet and create a new network with the name "CS6290" and RPC URL "http://192.168.43.142:12537", and then the address is automatically calculated and shown in the wallet, which is "net6290:aakpxuj09wfycdhc9gfz7yzkcgvm8kmrnyjt430w9p". At this moment, the other two machines can also connect to the network in Fluent Wallet and get their addresses.

Step 4: Start mining. Then we stop the boot node, and set "mining_author" to the address mentioned above and "mining_type" to "cpu" to enable mining using the computational power of the CPU, see Fig.23. Then we can run the boot node for the second time to start mining, and the mining reward can be seen via Fluent Wallet and ConfluxScan.

Step 5: Run the new nodes. The configuration of new nodes is similar to that of the boot node, except for one thing: set the "bootnodes" parameter. The parameter consists of two components which are node ID and IP address/port and link with "@". The node ID can be retrieved from the "log" of the boot node "Self node id: 0xf29274bd172655f79571eb4753b12909bf00d8d1cef4a7cfcc5

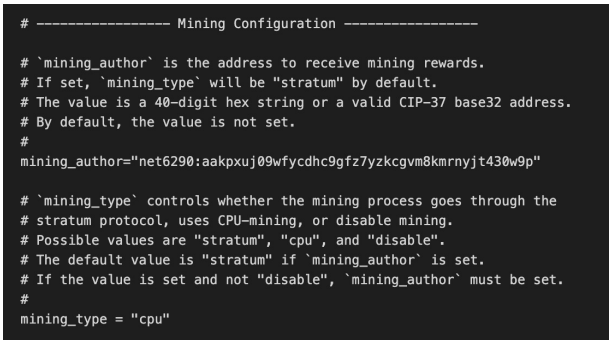

Fig. 23. Mining configuration

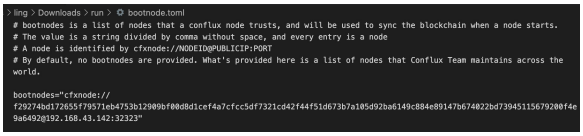

Fig. 24. Bootnodes configuration

Fig. 25. Trusted nodes

df7321cd42f44f51d673b7a105d92ba6149c884e89147b674022bd73945115679200f4e9a6492" removing prefix "0x". The IP address/port is the one mentioned above "192.168.43.142:32323". See Fig.24. Up to now, we can run the new nodes with the same command.

Step 6: Check node status. The newly-entered nodes will first be considered "untrusted" and converted to "trusted" after joining for a certain period of time, see Fig.25. This time interval can be customized by setting the "node_table_promotion_timeout_s" parameter. The two node lists can be found under "blockchain_data-net_config" directory.

C. Summary

During the process, we met several problems and managed to solve most of them by reading the source codes and discuss with Conflux team.

- We set our nodes in the same LAN so that the public IPs are actually LAN IPs in order to ensure RPC connections, e.g., wallet connection. The nodes cannot handle the forwarding of router NAT, thus some "best effort" improvements are implemented in node P2P ports, but not in RPCs. However, this only enables node connection with high possibility, depending on whether the router supports control protocols such as UPnP.
- The "min_phase_change_normal_peer_count" should be set to "1", otherwise the network will never enter the normal phase, because there are not enough boot nodes in the networks. The default value is "3", however, it's neither mentioned in the technical documents nor clarified in the code comments.
- We also find a weird problem that simply selecting or scrolling the output command line will make the thread stuck in Windows 10. This problem costs us a lot of time because the new nodes have several initialization works to do before synchronization.

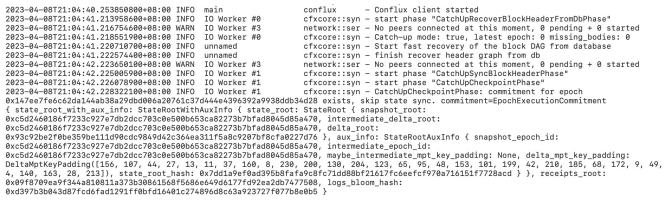

Fig. 26. Checkpoint

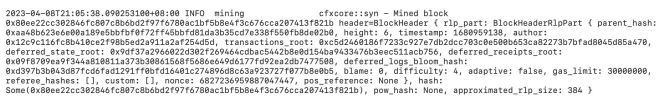

Fig. 27. Mined block

We also gain several interesting findings during the process and by reading the local data, which indeed deepen our understanding of the protocol.

- When the new nodes synchronize with the boot node, we notice that they are not synchronizing from the very beginning, but from a certain epoch that is close to the current height. This is because Conflux has a checkpoint mechanism to save local storage space to set checkpoints at every certain interval height, see Fig.26. For a full node, for the blocks before the nearest checkpoint, it will only store the block headers instead of the whole blocks.
- We also observe the process in which a node mines a new block, see Fig.27. Aside from the conventional fields like mining author or transaction root hash, due to the Tree-Graph structure, the new block has both parent hash and referee hash.
- We find special configurations related to the new features. For instance, as Fig.28 shows, Conflux protocol has two chain IDs, one for Core, and one for eSpace. And the hard fork activation time is set to "hydra_transition_number = 92060600" or "hydra_transition_height = 36935000", depending on the one that reaches earlier.

VI. Conclusion

This report explores Conflux, its consensus protocol, and novel features implemented following recent hard forks. The

```
# The chain ID of Conflux Network (Conflux space)
# 1 for testnet
# 1029 for Mainnet (Hydra)
#
chain_id = 6290

# The EVM chain ID of Conflux Network (EVM space)
# 1030 for Mainnet (Hydra)
#
evm_chain_id = 6291
hydra_transition_number = 92060600
hydra_transition_height = 36935000
cip43_init_end_number = 92751800
pos_reference_enable_height = 37400000
```

Fig. 28. New features

study involves reviewing papers on Conflux's system design and consensus algorithm, discussing two CIPs, testing basic operations on the Conflux Testnet, and running an independent Conflux chain for further investigation.

The process provides us with a deeper understanding of blockchain as well as Conflux protocol. Our testing and implementation results are not fully satisfying, however, we analyzed the methods and results, and gained a series of valuable knowledge and experiences, in both theoretical and engineering manner.

However, during this process, we also come up with several questions and directions about the protocol improvement.

- One of the motivations in the paper [2] is to deal with liveness attack, and the solution is to propose a Greedy Heaviest Adaptive SubTree (GHAST) mechanism by combining GHOST and structured GHOST. However, this is specially designed for PoW chain. The newly-introduced PoS chain can also lose liveness under 34% attack of staking.
- The PoS finality mechanism can help protect the assets and transactions before potential 51% attacks, however, cannot thoroughly and perfectly solve the problem of 51% attacks. Consider the situation of a 51% attack on PoW chain, the PoS chain now loses liveness because it's hard for the committee to achieve consensus and vote for a confirmed block, and the subsequent blockchain only has a PoW chain, with computational power less than 49% of previous. Consequently, it becomes much easier for another attacker to implement another 51% attack with less cost.
- Inspired by the previous point, it's interesting to take "sequential attacks" into consideration, which is similar to the core idea of liveness attacks. For instance, attackers with 34% staking tokens can not only disrupt the order of PoS committee, but also manipulate the price of CFX in outside exchanges to reduce the number of miners and total computational power of the network, which further makes it easier and more economical to implement a 51% attack.
- For blockchains using a Directed Acyclic Graph (DAG) structure, more communications among peers are required to maintain the DAGs and synchronization. From the engineering perspective, it means more resources are consumed, e.g., bandwidth and storage, which acts as a drawback considering the throughput and the cost for miners to operate a node. It's an open question to further explore other DAG-based solutions.

REFERENCES

[1] C. Li, P. Li, D. Zhou, W. Xu, F. Long, and A. Yao, "Scaling nakamoto consensus to thousands of transactions per second," 2018.

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

[3] ConFi-Conflux, "Conflux v2.0.0-fix network hard-fork upgrade announcement," 2022. [Online]. Available: https://forum.conflux.fun/t/conflux-v2-0-0-fix-network-hardfork-upgrade-announcement/13473

[4] F. Long and C. Li, "Cip-43: Introduce finality via voting among staked," 2021. [Online]. Available: https://github.com/Conflux-Chain/CIPs/blob/master/CIPs/cip-43.md

[5] Z. Voell, "Ethereum classic hit by third 51% attack in a month," 2020. [Online]. Available: https://www.coindesk.com/markets/2020/08/29/ethereum-classic-hit-by-third-51-attack-in-a-month/

[6] K. Reynolds, "Privacy coin grin is victim of 51% attack," 2020. [Online]. Available: https://www.coindesk.com/markets/2020/11/08/privacy-coin-grin-is-victim-of-51-attack/

[7] J. Redman, "Privacy coin verge suffers third 51% attack, analysis shows 200 days of xvg transactions erased," 2021. [Online]. Available: https://news.bitcoin.com/privacy-coin-verge-third-51-attack-200-days-xvg-transactions-erased/

[8] F. Long and C. Li, "Cip-90: A space fully evm compatible," 2022. [Online]. Available: https://github.com/Conflux-Chain/CIPs/blob/master/CIPs/cip-90.md

[9] G. Wood, "Ethereum yellow paper: a formal specification of ethereum, a programmable blockchain," 2022. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[10] C. C. Community, "Differences between cvm and evm," 2021. [Online]. Available: https://zhuanlan.zhihu.com/p/446509823

[11] PolygonLabs, "Polygon wiki," 2023. [Online]. Available: https://wiki.polygon.technology/

[12] OffchainLabs, "Arbitrum documentation center," 2023. [Online]. Available: https://developer.arbitrum.io/

[13] MatterLabs, "zksync era docs," 2023. [Online]. Available: https://era.zksync.io/docs/

[14] C. Li, "Cip-72: Accept ethereum transaction signature," 2021. [Online]. Available: https://github.com/Conflux-Chain/CIPs/blob/master/CIPs/cip-72.md

[15] C. Li and F. Long, "Cip-80: Ethereum compatible signature recover," 2021. [Online]. Available: https://github.com/Conflux-Chain/CIPs/blob/master/CIPs/cip-80.md

[16] ConfluxDocs, "Crossspacecall," 2023. [Online]. Available: http://doc.confluxnetwork.org/docs/core/learn/core-space-basics/internal-contracts/crossSpaceCall

[17] ConfluxDoc, "Evm compatibility," 2023. [Online]. Available: http://doc.confluxnetwork.org/docs/espace/build/evm-compatibility

[18] C. Network, "Exploring conflux's ecosystem," 2023. [Online]. Available: https://confluxnetwork.medium.com/exploring-confluxs-ecosystem-a061ac756d56

[19] S. Malwa, "Conflux network to build blockchain-based sim cards in partnership with china telecom," 2023. [Online]. Available: https://www.coindesk.com/tech/2023/02/15/conflux-network-to-build-blockchain-based-sim-cards-with-china-telecom/

[20] ConfluxNetwork, "Conflux developer portal," 2023. [Online]. Available: https://doc.confluxnetwork.org/

[21] "espacetestingcontract," 2023. [Online]. Available: https://evmtestnet.confluxscan.net/address/0x9624bd1e6547ce5d53d09d336f87fd19f67fb18d?tab=contract-viewer