2019 IEEE International Conference on Blockchain (Blockchain)

Enhancing Blockchain Traceability with DAG-based Tokens

Hiroki Watanabe, Tatsuro Ishida, Shigenori Ohashi,
Shigeru Fujimura, Atsushi Nakadaira, Kota Hidaka
NTT Service Evolution Labs., NTT Corp.
hiroki.watanabe.eh@hco.ntt.co.jp

Jay Kishigami
Muroran Institute of Technology
jay@csse.muroran-it.ac.jp

***Abstract*—Blockchain technology has positively impacted traceability across various industries such as logistics and shipping. By overcoming the problem of data silos among organizations and concentration of authority, this innovative approach allows a decentralized and tamper-resistant ledger to be built. On the other hand, the use of non-fungible tokens, a new concept for projecting digital and physical goods on a blockchain, is expected to be able to represent complicated operations in traceability systems. However, the existing token design lacks an efficient model for retrieval of past histories. The most popular way to retrieve past information on tokens is to use blockchain explorers, which are centralized trusted parties. Otherwise, so far there is only a naive way to examine all block headers, and it takes a vast amount of time. Our preliminary study shows that the processing time for retrieving all histories of 1,030 tokens on the Ethereum mainnet takes approximately 57 minutes. We propose a new token design based on a directed acyclic graph (DAG) that allows token histories to be efficiently explored without examining the whole blockchain. The model associates each state of the token from past to latest and covers expressions of relationships between tokens, such as merge and split. Our implementation and evaluation reveal that the DAG-based token design significantly improves the efficiency of exploring token transfer histories. The results show that our method can complete the exploration for 1,030 tokens in a matter of seconds and maintains performance over the long term even if the blockchain grows.**

I. Introduction

Blockchain technology has brought advances in terms of traceability and transparency to various industries. Supply-chain management for the shipping and logistics industries, a practical example of blockchain technology, benefits from this distributed ledger being updated and validated reliably. The food tracking system introduced by Walmart and its partners has made it possible to guarantee the production origin and quality of food and has reduced the time needed to trace the source of food from 7 days to 2.2 seconds [1]. The blockchain-based traceability system does not require users to place trust in a centralized system, which enables them to follow the distribution process with a decentralized approach while preventing fraud, corruption, tampering, and counterfeiting. In verifications using smart contracts, the reliability of an asset is confirmed in an autonomous and distributed manner; thus, companies can reduce the costs of both fraud and compliance. This powerful distributed tracking system reduces dependency on obsolete paperwork in all situations where traceability is required and shortens the time required for the whole workflow.

On the other hand, even more standardized and safer format specifications are required in data collaborations that take place on a global scale. Here, Ethereum has an advanced smart contract execution environment, and the token-related data format is being standardized. ERC-20 [2] provides standardized interfaces to access information on a token for wallet and tracking services, which include operations such as secure transfer of tokens and confirmation of balances. In contrast to the "fungible" ERC-20 token, the "non-fungible" token standard ERC-721 [3] is an identifiable token reflecting digital goods or physical goods. In this context, a proposal has been made to digitally tokenize physical goods and resources in a supply chain system [4].

In a token-based traceability system, everyone needs to be able to easily confirm the history of the circulation of tokens related to each product, which should ideally not involve a trusted third party. However, the traditional way to retrieve histories on a blockchain is to use a blockchain explorer service. This way requires the user to trust a centralized index database and to make an effort to follow the APIs provided by the explorer. Another way is to look directly at block data for the blockchain that the user owns. This naive way requires a lot time to search all the blocks; if the search space is large, it takes a huge amount of time. Although the existing token standards focus on secure input and interface design, they do not consider an efficient way of conducting history searches.

In this paper, we develop a method for efficient and trustless token state tracking. In contrast to existing studies focusing on input, such as how to project the product origin and distribution process onto a blockchain, the goal of our work is to enable flexible and efficient output related to tokens. Our tracking method allows changes in state of more than a thousand tokens to be captured instantly, with preserving the principle of self-validation of the blockchain. In particular, we propose a directed acyclic graph (DAG) based history management, which is also used for a distributed version control system. Our core idea is to incorporate the concept of DAG into the token design model. This model not only makes tracking more efficient; it also facilitates tracking of various token histories that include merges, splits, branches, etc.; i.e., it meets the requirements for tracking complex use-cases like supply chains. The main contributions of this paper are as follows.

- We propose a DAG-based token design which allows token

978-1-7281-4693-5/19/$31.00 ©2019 IEEE
DOI 10.1109/Blockchain.2019.00036

histories to be efficiently tracked in traceability systems.
- The token design supports flexible historical representations such as token merging, splitting, and forking.
- We implemented our token design and evaluated the processing time of token history retrieval.

The rest of this paper is structured as follows. Section II introduces the previously published literature related to our work. Section III presents the concept of the DAG-based token and contains a detailed description of the design. The experiments comparing our method with the existing ones are described in Section IV. Finally, conclusions and future prospects are presented in Section V.

II. Related Work

A. Challenges facing blockchain-based traceability systems

Several solutions have been proposed to enhance traceability using blockchain technology in order to comply with legal obligations, streamline inventory management, guarantee product quality, and counter fraud and more [4]–[10]. Industry has performed proof-of-concept demonstrations of typical supply chain operations, and some companies and organizations have launched traceability services for commercial use. However, considerable gaps have been found in the integration of supply chains using blockchain technology. One of the challenges is how to link physical products and digital products [5]. RFIDs, barcodes, and QR codes can be used to provide products with digital identifiers on the supply chain network, but these virtual IDs are physically replicable. For instance, Toyoda et al. propose a mechanism to detect replications of false physical tags by managing EPC information included in the RFID in association with information on the producer and owner of the product by using Ethereum smart contracts [6]. Abeyratne et al. point out that such digital profiles need to be continually updated through manual or automated systems [5].

Assuming that physical products can be properly projected to digital representations on a blockchain, the next problem is how to manage the digital representations while ensuring reliability, neutrality, and efficiency. Lu et al. state that the structural design of a smart contract affects not only performance but also updatability and overall system adaptability [7]. For example, if you move the logic to a blockchain, the reliability that the blockchain provides enable you to use it as a computing platform, but it may be difficult to renew your legal agreements or renew a smart contract for new regulations. This means that better standards and methodologies for development are needed for a blockchain-based traceability system. It is considered that a modeling approach can improve this state of affairs. For instance, Kim and Laskowski propose to introduce an ontology-based data model for traceability into the design of smart contracts [8]. The contracts based on the ontology support several typical supply chain operations such as production and consumption.

Meanwhile, models aimed at better data standards, standardized interfaces, dispersibility, and safety are being mainly discussed outside the traceability context. The overheating of the ICO (Initial Coin Offering) market in 2016 opened up a flood of competing token specifications. To address the problem of standardization, ERC (Ethereum Request for Comment) defines a common token format. The specification defines a list of common rules which Ethereum tokens should follow in the Ethereum ecosystem. By referring to the ERC-20 standards, developers can accurately determine the interactions between the tokens. These rules indicate how to transfer tokens between addresses and how to access data within each token. In contrast to the fungible ERC-20, ERC-721, a non-fungible token standard, has been proposed to meet that demand. Its purpose is to create a standardized interface for creating and trading identifiable tokens reflecting digital goods or physical goods.

Westerkamp el at. considered the distribution of products whose shapes may change and introduced the concept of tokens into the traceability system [4]. Supply chains continue to expand globally, and the design of the contract structure is increasingly difficult. A new token design that incorporates the badge concept in the ERC-721 token would allow more complex use cases. Token merges, splits, and forks can be used as expressions of the physical products assembly and formulation, division of rights, etc. beyond the scope of tracking a single item. However, the above studies, including [4], make no mention of how to efficiently search the tracking history.

B. Exploration methods for token tracking

A standardized interface allows a more efficient search of the transaction history of the token by linking with a third-party explorer. Etherscan[1] is an explorer for the Ethereum blockchain, and it provides advanced search and analysis platforms. Additionally, Etherscan can search standardized ERC-721 tokens with the contract address and its token ID as the key. This kind of explorer service offers efficient searches, but there are two concerns. One concern is that it is not a completely trustless architecture; every user needs to place trust in the service operators. Another one arises when we use a closed and isolated consortium-type blockchain. The cost to maintain the explorer service would be huge, because it must always monitor and analyze the blockchain; this may be an obstacle to introducing a blockchain system.

Barring an explorer service like the above, a naive way of searching the token history is to look through all past blocks. In this method, the time spent searching linearly increases as the search range expands. In Ethereum, this problem is alleviated to some extent by the mechanism of *event and logs* [11]. Ethereum smart contracts can emit events and write logs to the blockchain when mining. If indexed parameters are specified in the logs, the smart contract can add them to a Bloom filter in the header of each block. The Bloom filter enables the searcher to detect whether or not a specific event log has occurred in each block in seconds. However, unless the searcher has clues like on which block the events are emitted,

[1]https://etherscan.io/

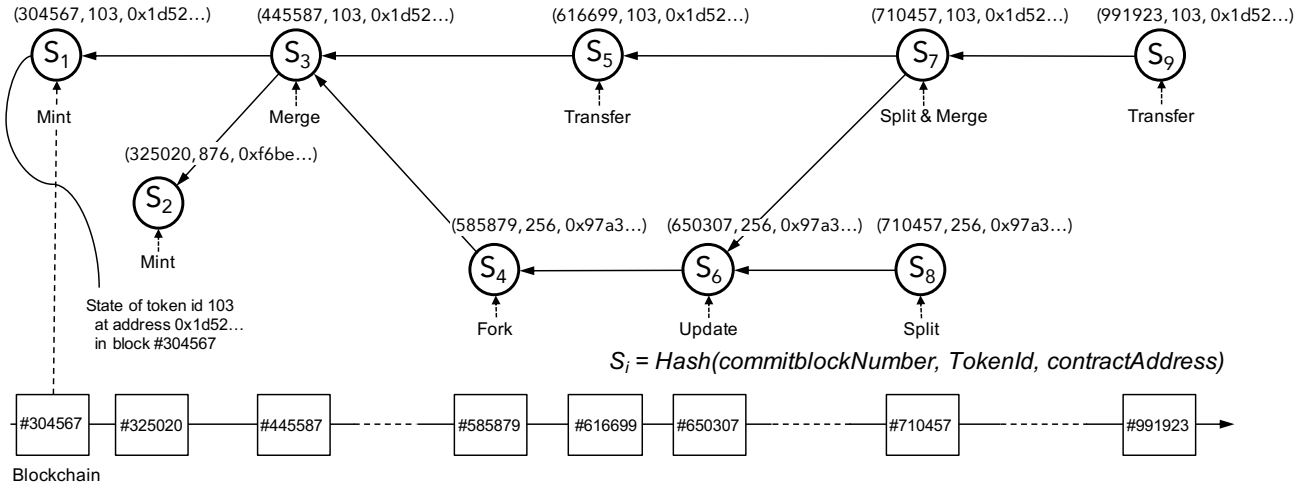

Fig. 1. State transition of token using DAG model

he or she would have to look through all the block headers; this would take a very long time.

III. Concept and design

We propose a new token design that makes a blockchain-based traceability system more efficient and flexible. Our system is based on the work of [4]; that is, the smart contract defines a token to which a good is projected, where recipes are implemented for different operations such as transferring, combining, and splitting tokens. The obvious difference is that our token design has a link to the state of the past tokens to improve traceability. Our core idea is to model token state transitions and to develop a DAG-based token design that can be searched directly from the blockchain. In computer science and mathematics, a DAG is defined as a graph that is directional and has no cycles connecting other nodes. The version control system Git[2] is a popular example of using DAGs; it provides a powerful Merkle DAG object model that obtains distributed changes to the file system tree in a distributed manner. A file for each version is expressed as an object, and the hash value of the file content is designated as the address for each object. A link to another object is embedded in the object, and the reference is updated every commit, which enables complex graph networks, such as those having file merges and forks, to be represented. This allows Git to implement a powerful file distribution strategy for any content. In addition, DAGs have been used in decentralized projects, such as IOTA [12], IPFS [13], etc.

The principle of our method is to model token state transitions by using a DAG. Figure 1 shows the basic concept and an example of our model. Each object represents the state of the token at the time of the block number, and each object has a link to the parent object, i.e., the past state of the token. A token object can have two or more parents at once (merge). Conversely, it is possible that one parent has two or more child tokens (fork or split). The object is represented by a three-tuple $(commitBlockNumber, tokenID, contractAddress)$, where $commitBlockNumber$ represents the block number on the blockchain when some change occurs to the token indicated by $tokenID, contractAddress$. For example, referring to Figure 1, although S_3 and S_5 indicate the same token, these are represented as a different node in our model because the ownership of the token is transferred by block number 616699. A significant insight is that the state of the token between the S_3 and S_5 links is identical to S_3. This means, when a searcher gets the histories of the related token ID, he/she only has to examine the state of the token indicated by the token object and does not have to search for all the blocks.

There are three types of object transition. One is "commit", which simply means that the user has made changes to the token, such as transfer of ownership and modification of metadata. The second operation is "merge", which indicates that multiple tokens are combined. For example, it can be used when manufacturing an item of furniture from and wood and metal fittings. The third one is "split" or "fork". Here, one can apply the concept of "split" to physical items and that of "fork" to digital content such as source code.

In the following, we first show the token design for embodying our model. Next, with regard to specific operations, we explain how operations such as commit, merge, and split should be implemented. It should be noted that we assume Ethereum (EVM) to be the foundation of the blockchain and use pseudo code based on Solidity to describe our algorithm.

A. Token Design

In our model, the state of the token at a certain block number is expressed as one object. It might seem that a smart contract has to instantiate a new token object every time a state transition of the token occurs. However, a blockchain can accumulate histories of past states; it is not necessary to create a new instance for each transition; instead, all that has to be created is the link to the past state in an actual

[2]https://git-scm.com/

implementation. Our DAG-based token inherits existing token designs such as ERC-721. Its important extension from ERC-721 is that it defines a parameter indicating the position of the parent token object; the parameter is stored as an array in the contract.

The position of the parent token object is specified by three parameters, $commitblockNumber$, $tokenId$, $contractAddress$. Accordingly, each token object has a unique identifier represented by such as a hash value:

$$Hash(commitBlockNumber, tokenId, contractAddress)$$

This value is referred to as a "commit hash" in the rest of this paper. We should note that all parameters within the token object are specified not by external entities, but by only the logic on the smart contract. Opcodes used within EVM include those that can refer to the context of the time of smart contract execution, such as block number $(0x43)$ and contract address $(0x30)$. According to the Solidity specification, the block number can be specified by $block.number$ and the address of the current contract can be specified by $address(this)$. These parameters are set by miners when the block is mined. Moreover, $tokenId$ is uniquely determined by the calculation of the smart contract. Consequently, the token object is specified autonomously, and even the administrator of the token can not selfishly rewrite it.

In addition to the parent token object information, we should note that the DAG-based token has a flag as a parameter indicating that it has been consumed. The concept of consumption of resources was introduced in [4]. We use this concept to design merge and split.

B. Commit

In our work, "commit" straightforwardly means completing the update of information on tokens. In a supply chain, the operations in which commits occur include simple transfers of ownership, records of audits, and certifications by certification authorities. These operations would be carried out by producers, suppliers, distributors, retailers, and end consumers; that is, commit records a trail of user operations on the token contract on the blockchain. The implementation should preferably call the commit function internally at the end of the function that performs each operation. An example implementation of the internal function is as follows:

```
function _commit(uint256 tokenId, bytes memory
    commitMessage) internal {
  commitBlockNumber[tokenId] = block.number;
  emit Commit(tokenId, commitMessage);
}
```

In the internal function $\_commit$, the block number at the time of mining is acquired and stored using the token ID as a key, which means that a link to the past state is automatically generated by the smart contract. Since the token ID and contract address are obvious to the searcher, the commit hash can be calculated from the block number. Ethereum full nodes have the whole transaction history; hence, the past states can be promptly acquired simply with the block number. The actual search procedure is described in section III-E.

C. Merge

The existing systems of traceability over multiple businesses lose tracking when products are combined and processed into a new product. In practice, it is often the case that different physical materials are combined. For example, in the manufacture of furniture, wood and metal fittings are shipped separately and are consumed in a plant that assembles them into furniture. This manufacturing process is expressed by combining tokens that are projected to goods. In general-purpose explorers, there is no way to track the binding between different ERC-721 tokens despite them being required in the context of the traceability system. The main reason for token tracking becoming impossible is that each token is managed with a different smart contract and is logically decoupled. Since a token after being merged does not hold any information of the original token, a user trying to trace the token state is forced to make excessive effort to link information between different smart contracts.

Our DAG-based token has a pointer to access the past state of the token, which is also possible over different tokens on different contracts. The following shows the simple logic of the proposed merge process:

```
function _merge(uint256 tokenId, address[] memory
    parentTokenAddrs, uint256[] memory
    parentTokenIds) internal returns (bool)
{
  bytes32 commitHash = getCommitHash(tokenId, block.
      number);

  //Store array of merged tokens
  Parents[commitHash].tokenIds = parentTokenIds;
  Parents[commitHash].tokenAddrs = parentTokenAddrs;

  for(uint i=0; i<_parentTokenAddrs.length; i++) {
    _consume(_parentTokenIds[i]);
  }
  _commit(tokenId, "Merge");
  return true;
}
```

The code expresses the merge process at a certain commitment point and links multiple parent token objects to a current token. The commitment point is represented by a commit hash, which also means the identifier of the position of the token objects. In addition, to prevent reuse after merging, the original token is consumed and never used again for inputs.

D. Split and Fork

In contrast to the merge operation, which combines and consumes multiple tokens and generates one token, we also consider the operation of splitting or forking one token into multiple tokens. Here, for example, batches of physical goods may be divided up and distributed as different tokens. Also, digital intangible goods could be forked to different products (e.g., source code or licenses). Thus, we incorporate the concept of split and fork in our model.

In the split operation, the original token is completely consumed, and a token with a different token ID is generated. This is mainly used for the dividing up physical goods. The

following internal function *_split* is called by the contract managing the original token:

```
function _split(uint256 tokenId, address[] memory
    childTokenAddrs, uint256[] memory childTokenIds)
     internal returns (bool) {
  bytes32 commitHash = getCommitHash(tokenId, block.
      number);

  Children[commitHash].tokenIds = childTokenIds;
  Children[commitHash].tokenAddrs = childTokenAddrs;

  _consume(tokenId);
  _commit(tokenId, "Split");
  return true;
}
```

As in the above example, the divided token can arbitrarily hold the information of the child token objects as information of the division destinations. On the other hand, not only the parent token but also the child tokens themselves should refer to the original parent tokens:

```
function _referTo(uint256 newTokenId, address
    parentTokenAddr, uint256 parentTokenId) internal
     returns (bool) {
  bytes32 commitHash = getCommitHash(_newTokenId,
      block.number);

  Parents[commitHash].tokenIds.push(_parentTokenId);
  Parents[commitHash].tokenAddrs.push(
      _parentTokenAddr);

  return true;
}
```

The internal function $\_referTo$ is called by the child tokens, i.e., the token of the split destination. It records the original token object before the split as the parent token.

In the fork process, the original token is not consumed, and the same content is allowed to generate a token as a secondary copy. This sort of process is common in intangible assets such as source code and intellectual property licenses. The $\_referTo$ function can be straightforwardly used for a fork operation in which a new token is created as a branch. Thus:

```
function _fork(uint256 newTokenId, address
    parentTokenAddr, uint256 parentTokenId) internal
     returns (bool) {

  _referTo(newTokenId, parentTokenAddr,
      parentTokenId);

  _commit(tokenId, "Fork");
  return true;
}
```

In the case of the fork operation, since no request for the original token occurs, the token contract of the fork destination calls the $\_fork$ function and unilaterally becomes associated with the original token.

E. Exploring tokens

Tracking of token state transitions is made possible by recursively searching every token object starting with the latest state. The number of search steps is proportional to the number of token objects (i.e., the number of commits). If the number of token objects is N, the search of all token objects can be completed in $O(N)$ steps.

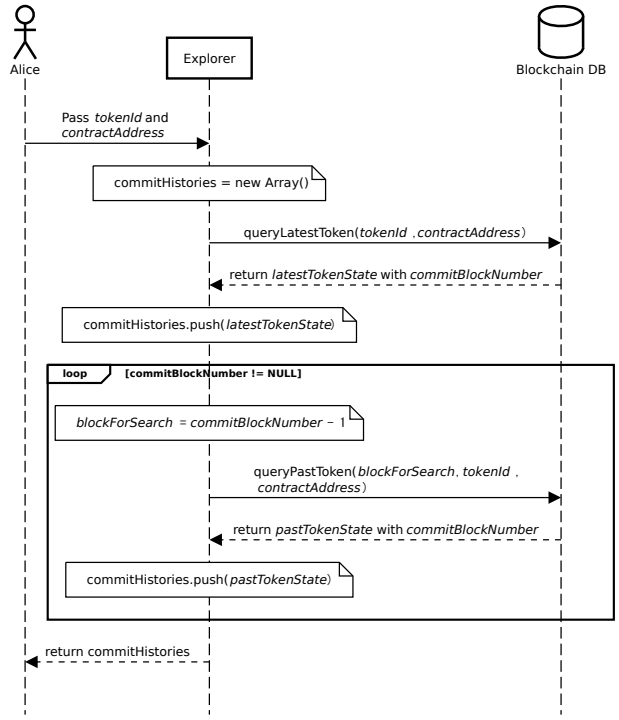

Fig. 2. Exploring all state histories of a DAG-based token

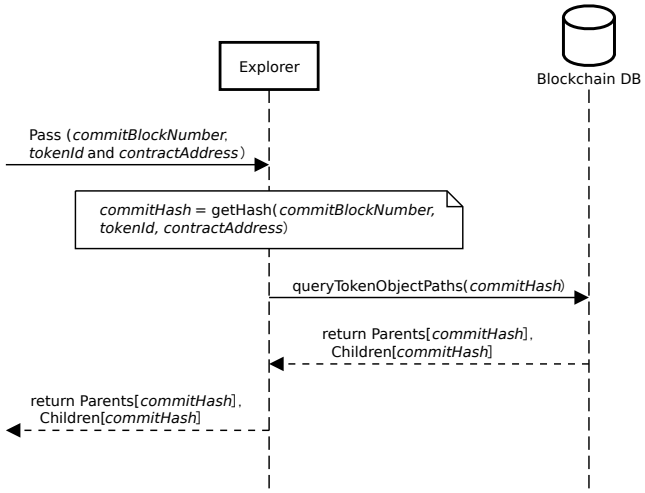

Fig. 3. Exploring paths to another token across different contracts using DAG

Figure 2 shows the sequence diagram of the tracking process for a token ID. First, the searcher Alice passes the token ID and its contract address to the explorer, and the explorer obtains the latest state of the token from the blockchain. The result includes the committed block number, which means that the explorer can see the token object (i.e., the 3-tuple, $commitBlockNumber$, $tokenID$, $contractAddress$). In the next inquiry, the explorer specifies the block number of the value decreased by 1 from the committed block number and

queries the past token state. This processing is recursively performed until a committed block number cannot be found.

This search process utilizes the link to the parent token object that is automatically formed by the blockchain mechanism of stacking blocks; that is, by looking into the previous block, if the token ID and contract address remain unchanged, the explorer can straightforwardly obtain the information on the parent token object. On the other hand, when a merge or split process is found, the explorer needs to know the paths on which to search across multiple smart contracts. Figure 3 shows how to obtain paths to other token objects (parents or children objects). As described in the previous section, the token stores the paths associated with a commit hash; thus, by using a commit hash, the explorer can obtain paths across different contracts.

IV. Analysis

To assess the proposed DAG-based token design, we conducted two experiments in which we measured the retrieval time of the token history, i.e., the processing time to obtain all histories of token state transitions. First, we measured it with certain ERC-721 tokens on the Ethereum mainnet, as a preliminary study to clarify the issues of the existing exploration method. Then, we measured it with the proposed DAG-based and ERC-721 tokens on our private Ethereum blockchain and compared their retrieval performances in detail. The private blockchain was built using Proof-of-Authority algorithm (Clique [15]) with four seconds block interval to allow faster block generation time, where each block would have the same data structure as that of the mainnet.

All measurements were performed on an Intel NUC 7i7DNKE running Ubuntu 16.04.5 LTS equipped with four cores, 1.90GHz Intel Core i7-8650U, and 32 GB memory. Our prototype was implemented as a Node.js application connected to the go-ethereum client (v.1.8.21).

A. Evaluation with ERC-721 tokens on Ethereum mainnet

The most popular non-fungible token standard is ERC-721. In this experiment, we measured the retrieval time of the transfer histories of ERC-721 tokens on the mainnet and analyzed how the performance of the existing exploration method changed as the block length increased.

Preparations

Our measurement target was the blockchain-based project, Decentraland [14]. This project provides a platform to tokenize the ownership of land in a VR space. It has created approximately 68,000 tokens (named "LAND") and has recorded more than 110,000 token transfers so far. The main reason we focused on the project is its openness; i.e., it is easy to retrieve the token data on the blockchain. In the Decentraland token, the issued event logs are all indexed by token ID; thus, any user can easily retrieve past event logs (Transfer, Approval, etc., standardized by ERC721) from the Ethereum mainnet blockchain. (Although the most popular ERC-721 project as of January 2019 is CryptoKitties, it does not have indexed fields in its event logs.)

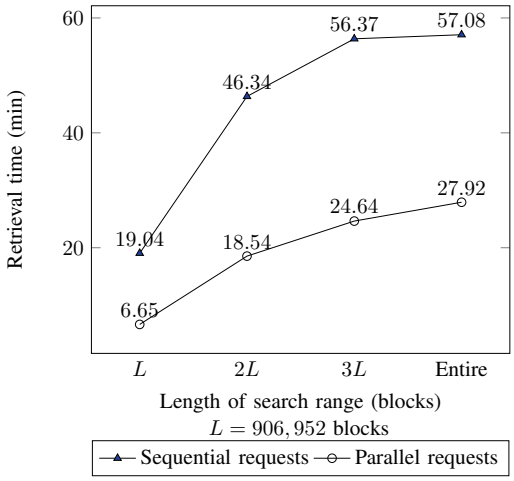

Fig. 4. Results of experiment with existing exploration method on mainnet

First, we retrieved information published on Etherscan, a block explorer service, and obtained 8000 LAND token IDs and the number of transfers (i.e., in our approach, the number of commits) associated with them. The number of transfers of each token was under 18 and followed an exponential distribution; thus, most of the tokens have been transferred only once or two times. Given the practical use case of the traceability system, token tracking would be used for a solid amount of tokens rather than for each token; therefore, in the situation depicted in the experiment, the user investigates the history of approximately 1,000 tokens. Next, we chose 1,030 tokens from the 8,000 LAND tokens. The number of the token transfers followed an exponential distribution from one to 18 token transfers, and the average number of transfers per token was 2.523 times.

Evaluation

As it stands, without a trusted explorer service like Etherscan, the most efficient way to search is to use an index field in Ethereum events and logs. According to the ERC-721 standard, an event is emitted and the log is recorded in a blockchain every time an operation like a transfer occurs. The log with the indexed field is stored as topic data in a Bloom filter in the header of each block; in this way, the search algorithm can efficiently judge whether or not the target value is included in all the transactions related to the block.

Figure 4 shows the measurement result for the 1,030 tokens in the above sort of search. The results are turnaround times from the start of the history search to completion in which all the transfer histories are returned. The experimental analysis compared results obtained under the following conditions.

Search range: We performed the experiments while changing the search range multiple times: the whole length of the blockchain that we downloaded was 7,147,700 blocks, i.e., that of the Ethereum mainnet as of January 2019. We analyzed the smart contract of the LAND token and found that the event

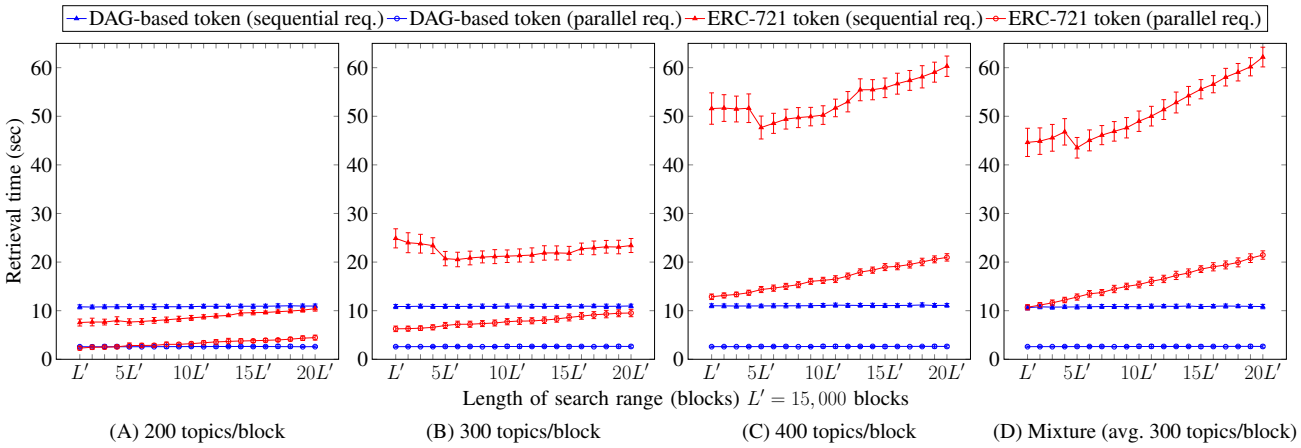

Fig. 5. Comparison of results of exploring DAG-based (blue plot) and ERC-721 tokens (red plot)

logs related to the 1,030 tokens were included in blocks after the 6,240,748th. Therefore, we defined the block length L (= 7,147,700 - 6,240,748 = 906,952 blocks) as the basic search range that contained all the logs we should measure. $2L$ and $3L$, which are also search ranges that have all the event logs, are two times and three times as long as the basic search range L. We also performed a measurement that did not specify a search range, which means that the explorer scanned the entire blockchain.

Type of request: The request for the go-ethereum client was performed in a sequential or a parallel manner. In the former manner, the explorer issues requests in series and the go-ethereum client process the 1,030 requests in a sequential manner. In the latter manner, the explorer issues 1,030 requests all at once and the client processes them in a parallel manner.

Results and Discussion

The results shown in Figure 4 revealed the following points. Overall, the longer the search range, the longer it takes for the history search to complete. Especially when the searcher does not have any clue to the search range, he or she has to look through all block headers over the whole range. Parallelization of the requests helps to reduce the amount of wasted time, but we consider its effect is not sufficient. This is because the results of the parallel processing should be regarded as maximum-effort values. In fact, we observed that the process consumed almost all of the CPU resources in the machine. Under worst-case conditions, developers and operators should assume that up to 57 minutes are needed for 1,030 tokens (see, the condition, *Sequential requests* and *Entire* in Figure 4).

B. Evaluation with proposed DAG-based tokens

The previous section described applying the existing search method to the existing tokens on the mainnet and clarified how inefficient the existing method is and how block length affects the retrieval time. Here, we conducted more detailed experiments comparing the efficiency of the proposed DAG-based token exploration with that of the existing ERC-721 token exploration.

In addition to block length, we added another viewpoint to study how performance is affected. We considered that the false positives in the Bloom filter influence the efficiency of retrieval. The Bloom filter in the block header helps users to detect whether the query exists in the block, but it algorithmically has a possibility of false positives, which would trigger unnecessary database queries. This wastes time and is an obstacle to real-time history analysis and system cooperation, that is, the larger the number of event logs (i.e., the amount of topic data) in the block, the higher the density of the Bloom filter and the higher the false positive rate.

Preparations

In contrast to the first experiment with the mainnet, this experiment used a private type of blockchain to look at the influence of the Bloom filter. The tests were performed using four private blockchains constructed with different event densities: low (200 topics/block), medium (300), high (400), and mixture (composed of blocks with different densities, average 300). These conditions were selected because the most recent blocks of Ethereum mainnet have around 300 event topics in them on average. The low, medium, and high densities were composed of blocks of uniform event density, while the mixture density had various blocks with event densities from 200 to 400; therefore, the mixture blockchain can be considered to be more similar to the environment of the Ethereum mainnet than the others. The conditions of the tokens to be measured were set to the same as the conditions of the LAND tokens used in the first experiment: here were 1,030 tokens, and the total number of transfers of them was 2,598 times. We should note that the block length of each blockchain was around 300,000 blocks, shorter than that of the Ethereum mainnet; hence, the search range was consequently shorter than in the previous experiment.

Evaluation

The DAG-based token histories were obtained using the exploration method described in section III-E. It should be noted that the method does not have to specify a search

range. Therefore, in the process of growing the blockchain, measurements were performed at several points determined according to the block length. The first measurement point was at the 15,000th block, and all transfer events were included within the 15,000 blocks. After that, measurements were performed every 15,000 blocks. Consequently, the basic search range is regarded as L' (= 15,000 blocks) and the ranges that $2L'...20L'$ indicate are two times...twenty times L'.

For comparison, we evaluated an ERC-721 token exploration method using the same blockchain we had built. The method catches all event logs as in the case of the first experiment in the mainnet. The block length of the basic search range was 15,000 blocks, in common with the DAG-based token's experiment. As with the previous experiments on the mainnet, we also compared the results of sequential and parallel request methods. Additionally, each measurement was performed 100 times and the average was calculated.

Results and Discussion

The results of all measurements are shown in the red and blue plots in Figure 5. The most important insight we can see from the results is that the retrieval time for the DAG-based tokens is almost the same for all measurements (see the blue plots in Figure 5). In other words, the method of searching DAG-based tokens is not affected by either the block length or the Bloom filter. The above facts are consistent with our understanding of the algorithm of DAG-based token exploration. In contrast, the results of the existing exploration method (red plots) indicate that the retrieval time increases as the block length grows, as was observed in the experiments using LAND tokens. We also found that the retrieval time becomes much longer as the density of a Bloom filter increases.

Additionally, we found that another factor, i.e., the heterogeneity of the blocks, degrades the performance of the existing exploration method. As can be seen from the Figure 5(B) and Figure 5(D), even if a blockchain has the same average topics per block (300 topics/block in case of the figures), the exploration on a blockchain consisting of blocks with different numbers of event topics is less efficient than the exploration on a blockchain with blocks having a uniform number. We consider that the randomness of the number of events contained in a block constitutes a limiting factor to parallel processing in the go-ethereum client; that is, the presence of blocks with high event density lowers the search performance. This fact would explain why the existing search method in the Ethereum mainnet is extremely inefficient; the blockchain of the mainnet is even more heterogeneous because of some blocks with high event density, which impedes the efficiency of the retrieval process.

It is worth emphasizing that the DAG-based tokens have an advantage in that retrieval performance does not deteriorate over the long term, even taking into account that a private-type blockchain was used in the experiments. Overall, these results suggest that the retrieval can be completed within several seconds even if the block length and event density increase.

V. Conclusion

We focused on the issue of efficiency for traceability of tokens on a blockchain. Our main idea is to introduce the DAG model into token designs like the ERC-721 standard. The DAG-based token is superior in two points: (i) it brings about a dramatic improvement in efficiency when exploring token state histories; (ii) it can represent the processing of several operations like merge, split, and fork, which are required in the context of the traceability system. Regarding point (i), we experimentally assessed its retrieval time and found that, when tracking 1,030 tokens, our method can keep the retrieval time to within a matter of seconds even if the block length and event density of the block grow. We would like to emphasize that our contribution provides an advanced traceability system with an analysis for efficient data circulation. We expect that the DAG-based token and its exploration method will remove some of the obstacles to and broaden the possibilities of distributed traceability systems. Specifically, real-time tracking using DAG-based tokens would facilitate cooperation with other systems such as an AI-based analysis platform.

References

[1] R. Miller, " Walmart is betting on the blockchain to improve food safety," 2018. [Online]. Available: https://techcrunch.com/2018/09/24/walmart-is-betting-on-the-blockchain-to-improve-food-safety/. [Accessed: 30- May- 2019].

[2] F. Vogelsteller and V. Buterin, "ERC-20 Token Standard," 2015. [Online]. Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md. [Accessed: 30- May- 2019].

[3] W. Entriken, D. Shirley, J. Evans, and N. Sachs, "ERC-721 Non-Fungible Token Standard," 2018. [Online]. Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md. [Accessed: 30- May- 2019].

[4] M. Westerkamp, F. Victor, and A. Kupper, "Blockchain-based Supply Chain Traceability: Token Recipes model Manufacturing Processes," in *2018 IEEE International Conference on Blockchain*, pp. 1595-1602, 2016.

[5] S.A. Abeyratne and R.P. Monfared, "Blockchain ready manufacturing supply chain using distributed ledger," *International Journal of Research in Engineering and Technology*, vol. 5, issue 9, pp. 1-10, 2016.

[6] K. Toyoda, P. Mathiopoulos, I. Sasase, and T. Ohtsuki, "A Novel Blockchain-Based Product Ownership Management System (POMS) for Anti-Counterfeits in The Post Supply Chain," *IEEE Access*, vol. 5, pp.17465 - 17477, 2017.

[7] Q. Lu and X. Xu, "Adaptable blockchain-based systems: A case study for product traceability," *IEEE Software*, vol. 34, issue 6, pp. 21-27, 2017.

[8] H. M. Kim and M. Laskowski, "Towards an Ontology-Driven Blockchain Design for Supply Chain Provenance," *SSRN Electronic Journal*, 2016.

[9] F. Tian, "A Supply Chain Traceability System for Food Safety Based on HACCP, Blockchain & Internet of Things," in *2017 International Conference on Service Systems and Service Management*, 2017.

[10] F. Tian, "An Agri-food Supply Chain Traceability System for China Based on RFID & Blockchain Technology," in *2016 International Conference on Service Systems and Service Management*, 2016.

[11] G. Wood. "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf. [Accessed: 30- May- 2019].

[12] S. Popov, "The Tangle," 2016. [Online]. Available: https://www.iota.org/research/academic-papers. [Accessed: 30- May- 2019].

[13] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3)," 2014. [Online]. Available: https://arxiv.org/pdf/1407.3561.pdf. [Accessed: 30- May- 2019].

[14] E. Ordano, A. Meilich, Y. Jardi, and M. Araoz, "Decentraland whitepaper," 2017. [Online]. Available: https://decentraland.org/whitepaper.pdf. [Accessed: 30- May- 2019].

[15] P. Szilágyi, "Clique PoA protocol & Rinkeby PoA testnet," 2017. [Online]. Available: https://github.com/ethereum/EIPs/issues/225. [Accessed: 30- May- 2018].