StreamNet: Enabling Large Scale DAG based Nakamoto Consensus through Streaming Graph Computing

***Abstract*—To achieve high throughtput in the POW based blockchain systems, a series of methods has been proposed, and DAG is one of the most active and promising field. We designed and implemented the StreamNet aiming to engineering a scalable and endurable DAG system. When attaching a new block in the DAG, only two tips are selected. One is the 'parent' tip whose definition is the same as in Conflux [17], another is using Markov Chain Monte Carlo (MCMC) technique by which the definition is the same as IOTA [24]. By leveraging the graph streaming property, we infer a pivotal chain along the path of each epoch in the graph, and a total order of the graph could be calculated without a centralized authority and high transaction validation speed will be achieved even if the DAG is growing.**

***Keywords*-Block chain, DAG**

I. Introduction

Ever since bitcoin [22] has been proposed, blockchain technology has been widely studied for 10 years. Extensive adoptions of blockchain technologies was seen in real world applications such as financial services with potential regulation challenges [20], [29], supply chains [14], [30], [2], health cares [3], [32] and IOT devices [5]. The core of blockchain technology resides in the consensus algorihtms applying to the real distrubuted computing world. Where computers can join and leave the network and these copmuters can cheat.

As the first protocol that can solve the so called Byzantine general's problem, bitcoin system suffers from the problem of low throughput with a transaction per second (TPS) of approximately 7, and long confirmation time (about an hour). As more and more machines joined the network, they are competing for the privileges to attach the block (miners) which result in huge waste of electric power. While sky rocketing fees are payed to make sure the transfers of money will be placed in the chain. On par, there are multiple proposals to solve the low transaction speed issue.

One method intends to solve the speed problem without changing the chain data structure, for instance, the bitcoin cash (BCH) fork of bitcoin (BTC) system tries to improve the throughput of the system by enlarging the data size of each block from 1 Mb to 4 Mb. To minimize the cost of POW, a proof of stake method POS [31] is proposed to make sure that only those who in the network can have the privilege to attach the block only if they have a large amount of token shares. Anohter idea targeting at utilizing the power in POW to do useful and meaningful tasks such as training machine learning models are also proposed [19]. In addition, inspired by the PBFT algorithm [4] and a set of its relateted variations, so called hybrid chain was proposed. The general idea is to use two step algorithm, the first step is to elect a commiette, the second step is collecting committee power to employ PBFT for consensus. Bitcoin-NG [9] is the early adoptor of this idea, which splits the blocks of bitcoin into two groups, one is for master election and another for regular transaction blocks. Honey-badger [21] is the system that firstly introducted the consensus commitee, it uses a predefined memebers to perform PBFT algorithm to reach consensus. The Byzcoin system [13] brought forth the idea of POW for the commitee election, and uses a variation of PBFT called collective signing for speed purposes. The Algorand [10] utilizes a random function to anonymously elect commetee and use this commitee to commit blocks, and the member of the commitee only have one chance to commit block. All these systems have one common feature, the split of layers of players in the network, which results in the complexity of the implementation of the system.

While aforemetioned methods are trying to avoid side chains, another thread of effort is put on using direct acyclic graph DAG to merge side chains. The first ever idea comes with growing the blockchain with trees instead of chains [27], which results in the well known GHOST protocol [28]. If one block links to ≥ 2 previous blocks, then the data structure grows like a DAG instead of tree [25], [26], [16]. There is a improvement of the GHOST based DAG algorithm which can achieve 6000 of TPS in reality [17]. Another set of methods tried to avoid finality of constructing a linear total order by introducing the probability of confirmation in the network [24], [6]. However, suffering from engineering issues, mainly due to the lack of transaction frquency and the growing complexity due to the network expansion. These system in reality are rely on centralized mehods to maintain their stability. Some of the side chain methods also borrows the idea of DAG, such as nano [15] and vite [18]

All of the current DAG systems used the idea of head counting method to infer main chains which does not consider the network structure. Dating back to the time Google uses PageRank [23] to sort importance of web page instead of number of references, and this method has achieved a huge susccess. And this method can also be utilized to help growing the DAG, in this paper, we use the Katz centrality metric [12]. Emerging social network research has introduced the method of streaming graph analysis [8], [11], [7] which deals with how to quickly maintain information on a temporally or spatially changing graph without traversing the whole graph. The main contribution of this paper is how to utilize the streaming graph analysis methods to bring the DAG systems into real decentralized, and stabilized growing system.

II. Basic Design

A. Data structure

The local state of a node in the StreamNet protocol is a direct acyclic graph (DAG) $G =< B, g, P, E >$. B is the set of blocks in G. $g \in G$ is the genesis block. For instance, vertex g in Figure 1 represents the Genesis block. P is a function that maps a block b to its parent block $P(b)$. Specially, $P(g) = \perp$. In Figure 1, parent relationships are denoted by solid edges. Note that there is always a parent edge from a block to its parent block (i.e., $\forall b \in B$, $b, P(b) >\in E$). E is the set of directly reference edges and parent edges in this graph. $e =< b, b^{'} >\in E$ is an edge from the block b to the block $b^{'}$, which means that b' happens before b. For example in Figure 1, vertex 1 represents the first block, which is referenced by the subsequent block 2, 3 and 4. Vertex 5 has two edges, one is the parent edge pointing to 3, another is reference edge pointing to 4. When a new block is not referenced, it is called a tip. For example, in Figure 1, block 6 is a tip. All blocks in the StreamNet protocol share a predefined deterministic hash function Hash that maps each block in B to a unique integer id . It satisfies that $\forall b \neq b'$, Hash$(b) \neq$ Hash(b').

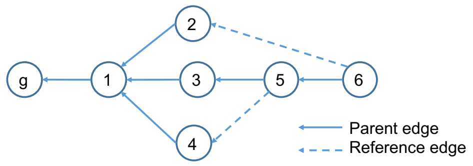

Figure 1. Example of the StreamNet data structure.

B. StreamNet Architecture

Figure 2 presents the architecture of StreamNet, it's consists of multiple StreamNet machines. Each StreamNet machine will grow its DAG locally, and will broadcast the changes using gossip protocol. Eventually, every machine will have a unified view of DAG. By calling total ordering algorithm, every machine can sort the DAG into a total order, and the data in each block can have a relative order regardless of their local upload time. Figure 3 shows the local architecture of StreamNet. In each StreamNet node, there will be a transaction pool acccepting the transactions from the HTTP API. And there will be a block generator to pack a certain amount of transactions into a block, it firstly find a parent and reference block to attach the new block to, based on the hash information of these two blocks and the meta data of the block itself, it will then perform the proof of work (POW) to calculate the nonce for the new block. Agorithm 1 summarize the server logic for a StreamNet node. In the algorithm, the way to find parent reference node is by $Pivot(G, g)$. And the way to find reference node is by calling $MCMC(G, g)$ which is the Markov Chain Monte Carlo (MCMC) random walk algorithm [24]. The two algorithms will be described in the later section.

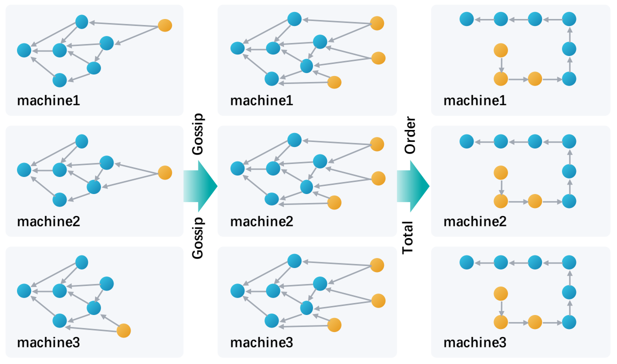

Figure 2. StreamNet architecture.

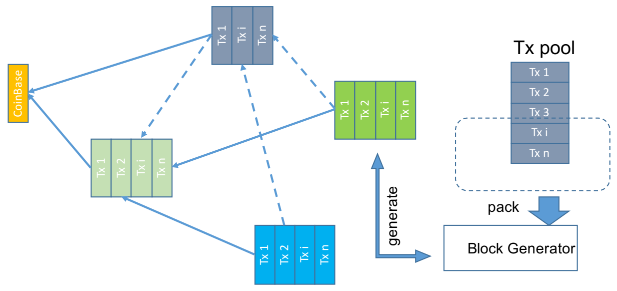

Figure 3. One node in StreamNet protocol.

C. Consensus protocol

Based on predefineid data structure, to present the StreamNet consensus algorithm, we firstly define several utility functions and notatons. BuildSubGraph() returns the sub graph by removing blocks and edges except the initial set of blocks. Score() presents the weight of blocks, each block achieves a score when attaching to the graph. Chain() returns the chain from the genesis block to a given block following only parent edges. Child() returns the set of child blocks of a given block. Sibling() returns the set of siblings of a given block. Subtree() returns the sub-tree of a given block in the parental tree. Before() returns the set of blocks that are immediately generated before a given block. Past() returns the set of blocks that are generated before a given block (but including the block itself). Figure 4 represents the definition of these utility functions.

Algorithm 1: StreamNet node main loop.

```
    Input: Graph $G =< B, g, P, E >$
1   while Node is running do
2       if Received $G' =< B', g, P', E' >$ then
3           $G'' \leftarrow < B \cup B', g, P \cup P', E \cup E' >$;
4           if $G \neq G''$ then
5               $G \leftarrow G''$ ;
6               Broadcase updated G to neighbors ;
7
8
9       if Generate block $b$ then
10          $a \leftarrow Pivot(G, g)$ ;
11          $r \leftarrow MCMC(G, g)$ ;
12          $G \leftarrow < B \cup b, g, P \cup < b, a >, E \cup < b, a > \cup < b, r >>$
13
14  end
```

Algorithm 2: MCMC(G, b).

```
    Input: The local state $G = < B, g, P, E >$ and a
           starting block $b \in B$
    Output: A random tip $t$
1   $t \leftarrow b$
2   do
3       for $b' \in Child(G, t)$ do
4           $P_{bb'} = \frac{e^{\alpha Score(G,b')}}{\Sigma_{z:z \to b} e^{\alpha Score(G,z)}}$
5       end
6       $t \leftarrow$ choose $b''$ by $P_{bb''}$
7   while Score(G,t) != 0;
8   return $t$ ;
```

$$\boxed{G = < B, g, P, E >}$$

$$Chain(G, b) = \begin{cases} g & \text{b = g} \\ Chain(G, P(b)) & \text{otherwise} \end{cases}$$

$$Child(G, b) = \{b' | P(b') = b\}$$

$$Sibling(G, b) = Child(G, P(b))$$

$$SubTree(G, b) = (U_{i \in Child(G,b)} Substree(G, i))U\{b\}$$

$$Before(G, b) = \{b' | b' \in B, < b, b' > \in E\}$$

$$Past(G, b) = (U_{i \in Before(G,b)} Past(G, i))U\{b\}$$

$$SubGraph(G, B') =< B', P', E' > |$$

$$\forall < b, b' > \in E', b \subset B' \& b' \subset B'$$

$$Score(G, b) = |SubTree(G, b)|$$

$$TotalOrder(G) = StreamNetOrder(G, Pivot(G, g))$$

Figure 4. The Definitions of Chain(), Child(), Sibling(), Subtree(), Before(), Past() and TotalOrder().

1) Pivot Chain Selection: The algorithm Algorithm 3 presents our pivot chain selection algorithm(i.e., the definition of $Pivot(G, b)$). Given a StreamNet state G, Pivot(G,g) returns the last block in the pivot chain starting from the genesis block g. The algorithm recursively advances to the child block whoes corresponding subtree has the largest number of blocks. Which is calculated by $Score(G, b)$ When there are multiple child blocks with the same score, the algorithm selects the child block with the largest block hash. The algorithm terminates until it reaches a tip. Each block in the pivot chain defines a epoch in the DAG, the nodes in DAG that satisfy Past(G,b) - Past(G,p) will belong to the epoch of block b. For example, in Figure 5, the pivot chain is $< g, 1, 3, 5, 6 >$, and the epoch of block 5 contains two blocks 4 and 5.

Algorithm 3: PIVOT(G, b).

```
    Input: The local state $G = < B, g, P, E >$ and a
           starting block $b \in B$
    Output: The last block in the pivot chain for the subtree
            of $b$ in $G$
1   do
2       $b' \longleftarrow$ Child($G$, $b$) ;
3       $tmpMaxScore \longleftarrow$ -1 ;
4       $tmpBlock \longleftarrow \perp$ ;
5       do
6           $score \longleftarrow$ Score($G$, $b'$) ;
7           if $score > tmpMaxScore$ || ($score$ = $tmpMaxScore$
            and Hash($b'$ ) < Hash(tmpBlock)) then
8               $tmpMaxScore \longleftarrow score$ ;
9               $tmpBlock \longleftarrow b'$ ;
10          end
11      while $b' \neq 0$;
12      $b \longleftarrow tmpBlock$ ;
13  while Child(G,b) != 0;
14  return $tmpBlock$ ;
```

2) Tip selection by MCMC: The tip selection method by using Monte Carlo Random Walk (MCMC) is as Algorithm 2 shows. Starting from the genesis, each random walk step will choose a child to jump to, and the probability of jumping from one block to the next block will be calculated using the formula in the algorithm. α in the formula is an constant that is used to scale the randomness of the MCMC function, the smaller it is, the more randomness will be in the MCMC function. The algorithnm returns until it finds a tip.

3) Total Order: The algorithm Algorithm 4 defines StreamNetOrder(), which corresponds to our block ordering algorithm. Given the local state G and a block a in the pivot chain, StreamNetOrder(G,a) returns the ordered list of all blocks that appear in or before the epoch of a. Using StreamNetOrder(), the total order of a local state G is defined as TotalOrder(G) in Figure 5. The algorithm in Figure 5 first recursively orders all blocks in previous epochs(i.e., the epoch of $P(a)$ and before). It then computes all blocks in the epoch of a as $B_\triangle$. It topologically sorts all blocks in $B_\triangle$ and appends it into the result list. The algorithm uses the unique hash to break ties.

Algorithm 4: StreamNetOrder(G, b).

```
Input:  The local state $G = < B, g, P, E >$ and a tip
        block $b \in B$
Output: The block list of total top order starting from
        Genesis block to the giving block $b$ in $G$
$L = \perp$
do
    $p \leftarrow$ Parent($G, b$) ;
    $B_\triangle \leftarrow$ Past($G$,$b$) - Past($G$,$p$) ;
    do
        $G' \leftarrow SubGraph(B_\triangle)$ ;
        $B'_\triangle \leftarrow$ {x || Before($G'$,$x$) = 0} ;
        Sort all blocks in $B'_\triangle$ in order as $b'_1, b'_2, ..., b'_k$
        such that $\forall 1 \leq i \leq j \leq k$, Hash($b'_i$) $\leq$ Hash($b'_j$) ;
        $L \leftarrow L + b'_1 + b'_2 + ... + b'_k$ ;
        $B_\triangle \leftarrow B_\triangle$ - $B'_\triangle$ ;
    while $B_\triangle \neq 0$;
    $b = p$ ;
while $b$ != $g$;
return $L$ ;
```

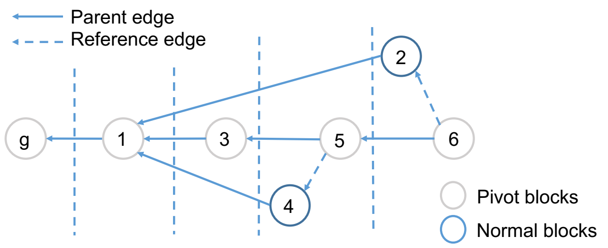

Figure 5. An example of total order calculation.

D. The UTXO model

In StreamNet, the transactions utilizes the unspent transaction out (UTXO) model, which is exactly the same as in Bitcoin. In the confirmation process, the user will call $TotalOrder$ to get the relative order of different blocks, and the conflict content of the block will be eliminated if the order of the block is later than the one conflicting with it in the total order. Figure 6 shows the example of storage of UTXO in StreamNet and how conifliction is resolved. Two blocks both referenced the same block with Alice having 5 tokens, and construct the new transaction out which representing the transfer of token to Bob and Jack respectively. However, after calling $totalOrder()$, the Bob transfer block preceeds the Jack transfer block, thus the later block will be discarded.

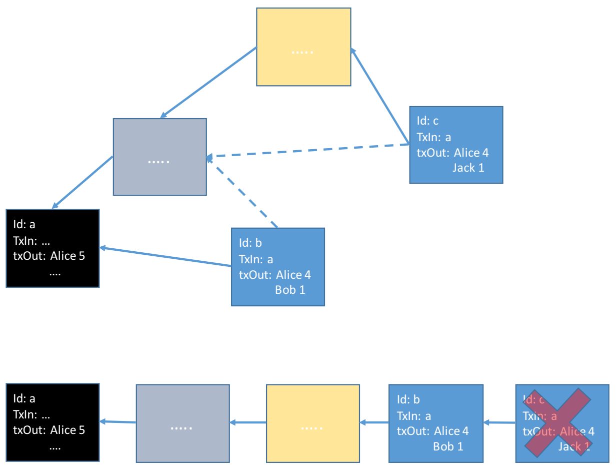

Figure 6. An example of UTXO.

E. Differences with other DAG protocols

Here, we mainly compare the difference of our protocol with two mainstream DAG based protocols, one is IOTA, another is Conflux.

1) IOTA: The major difference with IOTA are in three points, firstly, the IOTA tip selection algorithm's two tips are all randomly choosen, and ours is one by random one deterministic, secondly, the IOTA consensus algorithm is not purely decentralized, it relies on a central coordinator to issue milestones for the entry point selection, and our algorithm does not dependent on such facility. Lastly, in IOTA, there is no concept of total order, and there are 3 ways to judge if a transaction is confirmed:

- The first way is that the common nodes covered by all the tips are considered to be fully confirmed;
- All transactions referenced by the milestone tip are confirmed.
- The third way is to use MCMC. Call N times to select a tip using the tip selection algorithm. If a block is referenced by this tip, its credibility is increased by 1. After N selections have been cited M times, then the credibility is M/N.

2) Conflux: The major difference with Conflux are in two points, firstly, Conflux will approve all tips in the DAG along with parent, which is much more complicated than our MCMC based two tip method. Secondly, Conflux total ordering algorithm advances from genesis block to the end while StreamNet advances in the reverse direction. This method is one of the major contribution to our streaming graph based optimizations, which will be discussed in the next chapter.

III. Optimization Methods

One of the biggest challenges to maintain the stability of DAG system is that, as the local data structure growing, the graph algorithms ($Pivot()$, $MCMC()$,

Table I
ANALYSIS OF GRAPH PROPERTIES CALCULATION

Graph Property	Algorithm used	Complexity	Tot
Score(G, b)	MCMC()	$O(\|B\|)$	$O(\|B\|*d)$
Score(G, b)	Pivot()	$O(\|B\|)$	$O(\|B\|*d)$
Past(G,b) - Past(G,p)	StreamNetOrder()	$O(\|B\|)$	$O(\|B\|*d)$
TopOrder(G, b)	StreamNetOrder()	$O(\|B\|)$	$O(\|B\|)$

$StreamNetOrder()$), relies on some of the graph properties that needs to be recalculated everytime these functions are called, which are very expensive. Table I list all the expensive graph properties that are called. Suppose the set of blocks in the DAG is represented by B, and the depth of the pivot chain is d. Then we give the analysis of complexity in the following way. $Score()$ relies on the calculation of $SubTree()$ which is dependent on the breath first search (BFS), and the average BFS complexity would be $O(|B|)$, consider the scenario of MCMC() and Pivot() which are advanced through the main chain, the complexity would be in total $O(|B|*d)$ in both of these two cases. The calculation of $Past()$ also relies on the BFS operator, in the StreamNetOrder() algorithm, the complexity would be accrued to $O(|B|*d)$. TopOrder() is used in sub-order ranking the blocks in the same epoch. It's the classical topological sorting problem, and the complexity in the StreamNetOrder() would be $O(|B|)$.

Considering new blocks are genereted and merged into the local data structure in a streaming way. The expensive graph properties could be maintained dynamically as the DAG grows. Such that the complexity of calculating these properties would be armortized to each time a new block is generated or merged. In the following sections, we will discuss how to design streaming algorithms to achieve this goal.

A. *Optimization of Score()*

In the optimized version, the DAG will have a map that keeps the score of each block. once there is a new generated/merged block, it will trigger the BFS based UpdateScore() algorithm to update the scores of the block in the map that are referenced by the new block. The skeleton of the UpdateScore() algorithm is as Algorithm 5 shows.

B. *Optimization of Past(G,b) - Past(G,p)*

We abbreviate the Past(G,b) - Past(G,p) as GetDiffSet(G,b,C) which is shown in the Algorithm 6. This algorithm is in essence a dual direction BFS algorithm. Starting from the block b, it will traverse its referenced blocks. Every time a new reference block b' is discovered, it will perform a backward BFS to 'look back' to see if itself is already covered by the b's parent block of p. If yes, b' would not be added to the forward BFS queue. To avoid the

Algorithm 5: UPDATESCORE(G, b).

```
Input: Graph $G$, Block $b$, Score map $S$
Output: Updated score map $S$
$Q = [b]$ ;
$visited = \{\}$ ;
while $Q != \emptyset$ do
    $b' = Q.pop()$ ;
    for $b'' \in Before(G, b')$ do
        if $b'' \notin visited \land b'' != \perp$ then
            $Q.append(b'')$ ;
            $visited.add(b'')$ ;

    end
    $S[b'] + +$ ;
end
return $S$ ;
```

complexity of the backward BFS, the previous calculated diff set will be added to the covered set C, which will be passed to GetDiffSet() as a parameter. To be more specific, when a backward BFS is performed, the blocks in C will not be added to the search queue. This backward search algorihtm is denoted as IsCovered() and described in detail in Algorithm 7.

Figure 7 shows the example of the GetDiffSet() method for block 5. it first perform forward BFS to find block 4 which does not have children, then it will be added to the diff set. 4 then move forward to 1, which have three childrens, if it detect 3 which is the parent of 5, it will stop searching prompty. If it continue searching on 2 or 4, these two blocks would not be added to the search queue, because they are already in the covered set.

Algorithm 6: GETDIFFSET(G, b, C).

```
Input: Graph $G$, Block $b$, covered block set $C$
Output: diff set $D \leftarrow Past(G, b) - Past(G, p)$
$D = \emptyset$ ;
$Q \leftarrow [b]$ ;
$visited = \{b\}$ ;
$p = Parent(G, b)$ ;
while $Q != \emptyset$ do
    $b' = Q.pop()$ ;
    for $b'' \in Before(G, b')$ do
        if $IsCovered(G, p, b'', C) \land b'' != \perp$ then
            $Q.append(b'')$ ;
            $visited.add(b'')$ ;

    end
    $D.add(b')$ ;
    $C.add(b')$ ;
end
return $D$ ;
```

C. *Optimization of TopOrder()*

The topological order is used in sorting the blocks in the same epoch. To get the toppological order,

Algorithm 7: IsCovered(G, p, b', C).

```
Input: Graph $G$, Block $b'$, parent $p$, covered block set $C$
Output: true if covered by parent, else false
1  $Q \leftarrow [b']$ ;
2  $visited = \{b\}$ ;
3  while $Q! = \emptyset$ do
4      $b''$ = $Q.pop()$ ;
5      for $t \in Child(G, b'')$ do
6          if $t = p$ then
7              return true ;
8          else if $t \notin visited \wedge t \notin C$ then
9              $Q.add(t)$ ;
10             $visited.add(t)$ ;
11
12     end
13 end
14 return $false$ ;
```

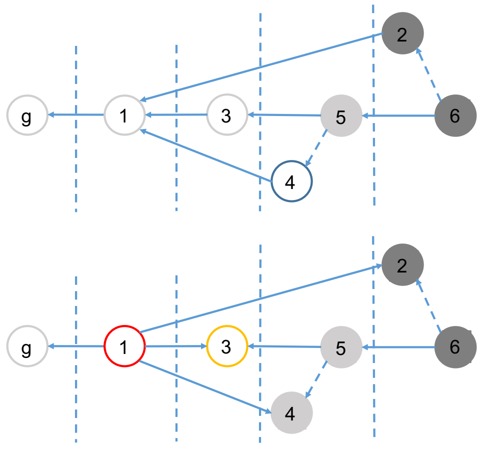

Figure 7. Example of the streaming get diff set method.

every time, there needs a top sort of the whole DAG from the scratch. However we can easily update the topological order when a new block is added or merged. The update rule is, when a new block is added, it's topological position will be $TopScore(G, b) \leftarrow min(TopScore(G, Parent(b)), TopScore(G, Reference(b)))$. This step can be done in $O(1)$

To summarize, the optimized streaming operators can achieve the performance improvement as Table II shows.

Table II
ANALYSIS OF GRAPH PROPERTIES CALCULATION

Graph Property	Algorithm used	Complexity	Tot
Score(G, b)	MCMC()	$O(\|B\|)$	$O(d)$
Score(G, b)	Pivot()	$O(\|B\|)$	$O(d)$
Past(G,b) - Past(G,p)	StreamNetOrder()	$O(\|B\|)$	$O(\|B\|)$
TopOrder(G, b)	StreamNetOrder()	$O(\|1\|)$	$O(\|1\|)$

IV. Experimental Results

A. Implementation

We have implemented the StreamNet based on the IOTA JAVA reference code (IRI) v1.5.5 [1].

- The features we have adopted from the IRI are:
  - the block header format;
  - gossip network;
  - trasnaction bundle, because of the existence of the bundle hash feature, StreamNet can support both the single transaction for a block and batched transactions as a bundle.
  - Sponge hash functions which is claimed to be quantum immune, in our experiment, the POW hardness is set to 8 which is the same as the testnet for IOTA.
- The features we have abandoned from the IRI are:
  - the iota's transaction logic inlcuding the ledger part;
  - the milestone issued by coordinators.
- The features we have modified based on the IRI is:
  - The tip selection method based on MCMC, since the tip selection on IRI has to find a milestone to start searching, we replace this with a block in the pivotal chain instead.
- The features we have added into the StreamNet are:
  - the consensus algorithms;
  - the UTXO logic which is stored in the signature part of the block header.

B. Environment Set Up

We have used the tencent cloud services, for each virtual machine, it includes a two core Intel(R) Xeon(R) CPU E5-26xx v4, with 3 Gb of memory size and 118Gb of disk size. The JAVA version is 1.8, we have deployed our service using docker and the docker version is 18.02.0-ce.

We have 7 machines set up, and they are connected into a gossip network in a cycle or in a clique. the requests will be distributed to these machines evenly. As for the data, we have created 10,000 accounts, with the genesis account having 100,000,000,000 tokens in the coinbase block. And we have issued 1,000,000 transactions in the network with 100 threads. Jmeter is utilized as the driver to issue the transactions and nginx is used to evenly distribute the requests to different nodes.

C. Results and Discussions

1) Single transaction test:

2) Bundle transaction test:

References

[1] "Iota reference implementation," *https://github.com/iotaledger/iri.*

[2] S. A. Abeyratne and R. P. Monfared, "Blockchain ready manufacturing supply chain using distributed ledger," 2016.

[3] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: Using blockchain for medical data access and permission management," in *Open and Big Data (OBD), International Conference on*. IEEE, 2016, pp. 25–30.

[4] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[5] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *Ieee Access*, vol. 4, pp. 2292–2303, 2016.

[6] A. Churyumov, "Byteball: A decentralized system for storage and transfer of value," *URL https://byteball. org/Byteball. pdf*, 2016.

[7] D. Ediger, R. McColl, J. Riedy, and D. A. Bader, "Stinger: High performance data structure for streaming graphs," in *2012 IEEE Conference on High Performance Extreme Computing*. IEEE, 2012, pp. 1–5.

[8] D. Ediger, J. Riedy, D. A. Bader, and H. Meyerhenke, "Tracking structure of streaming social networks," in *2011 IEEE International Parallel & Distributed Processing Symposium Workshops and PhD Forum*. IEEE, 2011, pp. 1691–1699.

[9] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol." in *NSDI*, 2016, pp. 45–59.

[10] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 51–68.

[11] O. Green, R. McColl, and D. Bader, "A fast algorithm for incremental betweenness centrality," in *Proceeding of SE/IEEE international conference on social computing (SocialCom)*, 2012, pp. 3–5.

[12] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, 1953.

[13] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 279–296.

[14] K. Korpela, J. Hallikas, and T. Dahlberg, "Digital supply chain transformation toward blockchain integration," in *proceedings of the 50th Hawaii international conference on system sciences*, 2017.

[15] C. LeMahieu, "Nano: A feeless distributed cryptocurrency network," *Nano [Online resource]. URL: https://nano. org/en/whitepaper (date of access: 24.03. 2018)*, 2018.

[16] Y. Lewenberg, Y. Sompolinsky, and A. Zohar, "Inclusive block chain protocols," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 528–547.

[17] C. Li, P. Li, W. Xu, F. Long, and A. C.-c. Yao, "Scaling nakamoto consensus to thousands of transactions per second," *arXiv preprint arXiv:1805.03870*, 2018.

[18] C. Liu, D. Wang, and M. Wu, "Vite: A high performance asynchronous decentralized application platform."

[19] S. Matthew and E. T. Nuco, "Aion: Enabling the decentralized internet," *Aion project yellow paper*, vol. 151, pp. 1–22, 2017.

[20] J. MICHAEL, A. COHN, and J. R. BUTCHER, "Blockchain technology," *The Journal*, 2018.

[21] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 31–42.

[22] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[23] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[24] S. Popov, "The tangle," *cit. on*, p. 131, 2016.

[25] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "Spectre: Serialization of proof-of-work events: confirming transactions via recursive elections," 2016.

[26] Y. Sompolinsky and A. Zohar, "Phantom, ghostdag."

[27] ——, "Accelerating bitcoins transaction processing," *Fast Money Grows on Trees, Not Chains*, 2013.

[28] ——, "Secure high-rate transaction processing in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 507–527.

[29] A. Tapscott and D. Tapscott, "How blockchain is changing finance," *Harvard Business Review*, vol. 1, 2017.

[30] F. Tian, "An agri-food supply chain traceability system for china based on rfid & blockchain technology," in *Service Systems and Service Management (ICSSSM), 2016 13th International Conference on*. IEEE, 2016, pp. 1–6.

[31] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[32] X. Yue, H. Wang, D. Jin, M. Li, and W. Jiang, "Healthcare data gateways: found healthcare intelligence on blockchain with novel privacy risk control," *Journal of medical systems*, vol. 40, no. 10, p. 218, 2016.