arXiv:1908.06405v2 [cs.DC] 7 Jun 2020

# StreamNet: A DAG System with Streaming Graph Computing

Zhaoming Yin[1],[3], Anbang Ruan[2], Ming Wei[2], Huafeng Li[3], Kai Yuan[3], Junqing Wang[3], Yahui Wang[3], Ming Ni[3], and Andrew Martin[4]

[1] StreamNet Chain LLC, Hangzhou, Zhejiang 310007, China,
stplaydog@gmail.com,
WWW home page: http://www.streamnet-chain.com/
[2] Octa Innovation, Beijing, 100036, China,
ar,wm@8lab.cn,
WWW home page: https://www.8lab.cn/
[3] TRIAS Lab, Hangzhou, Zhejiang, 310008, China,
lhf,yuankai,wangjunqing,wangyahui,ming.ni@trias.one,
WWW home page: https://www.trias.one/
[4] University of Oxford, Oxford, OX1 4BH, England,
andrew.martin@cs.ox.ac.uk,
WWW home page: https://www.cs.ox.ac.uk/people/andrew.martin/

**Abstract.** To achieve high throughput in the POW based blockchain systems, researchers proposed a series of methods, and DAG is one of the most active and promising fields. We designed and implemented the StreamNet, aiming to engineer 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 [1]; another is using Markov Chain Monte Carlo (MCMC) technique by which the definition is the same as IOTA [2]. 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. To scale up, we leveraged the graph streaming property; high transaction validation speed will be achieved even if the DAG is growing. To scale out, we designed the 'direct signal' gossip protocol to help disseminate block updates in the network, such that messages can be passed in the network more efficiently. We implemented our system based on IOTA's reference code (IRI) and ran comprehensive experiments over the different sizes of clusters of multiple network topologies.

**Keywords:** block chain, graph theory, consensus algorithm

## 1 Introduction

Since bitcoin [3] has been proposed, blockchain technology has been studied for 10 years. Extensive adoptions of blockchain technologies was seen in real-world applications such as financial services with potential regulation challenges [4,5], supply chains [6,7,8], health cares [9,10] and IoT devices [11]. The core of

blockchain technology depends on the consensus algorithms applying to the open distributed computing world. Where computers can join and leave the network, and these computers can cheat.

As the first protocol that can solve the so-called Byzantine general problem, the bitcoin system suffers from a low transaction rate with a transaction per second (TPS) of approximately 7, and long confirmation time (about an hour). As more machines joined the network, they are competing for the privileges to attach the block (miners), which results in a massive waste of electric power. While skyrocketing fees are paid 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, segregated witness [12] or off-chain technologies such as lightning network [13] or plasma [14]. Another hard fork way changed the bitcoin protocol, such as the bitcoin cash 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 computational cost of POW, multiple organizations have proposed a series of proof of stake method (POS) [15,16,17,18,19] to make sure that those who have the privilege to attach the block proportional to their token shares. Another idea targeting at utilizing the power in POW to do useful and meaningful tasks such as training machine learning models are also proposed [20]. Besides, inspired by the PBFT algorithm [21] and a set of related variations, the so-called hybrid (or consortium) chain was proposed. The general idea is to use a two-step algorithm; the first step is to elect a committee; the second step is collecting committee power to employ PBFT for consensus. Bitcoin-NG [22] is the early adopter of this idea, which splits the blocks of bitcoin into two groups: for master election and another for regular transaction blocks. Honey-badger [23] is the system that first introduced the consensus committee; it uses predefined members to perform the PBFT algorithm to reach consensus. The Byzcoin system [24] brought forth the idea of POW for the committee election and uses a variation of PBFT called collective signing for speed purposes. The Algorand [25] utilizes a random function to elect a committee and use this committee to commit blocks anonymously, and the member of the committee only has one chance to commit block. Other popular systems include Ripple [26], Stellar [27] and COSMOS [28] etc. All these systems have one common feature, the split of layers of players in the network, which results in the implementation complexity. While the methods above are aiming to avoid side chains, another thread of effort is put on using a direct acyclic graph(DAG) to merge side chains. The first-ever idea comes with growing the blockchain with trees instead of chains [29], which results in the well-known GHOST protocol [30]. If one block links to $\geq 2$ previous blocks, then the data structure grows like a DAG instead of tree [31], SPECTRE [32] and PHANTOM [33] are such type of systems. Byteball [34] is the system that constructs the main chain, and leverage this main chain to help infer the total order, nonetheless, the selection of the main chain is dependent on a role called to witness, which is purely centralized. Conflux is

an improvement of the GHOST based DAG algorithm, which also utilizes the pivotal (main) chain without the introduction of witness and claim to achieve 6000 of TPS in reality [1]. IOTA tried to avoid the finality of constructing a linear total order by introducing the probabilistic confirmation in the network [2]. As mentioned earlier, the systems are permissionless chains; in the permission chains, DAG technology is also applied. HashGraph [35] is the system that utilizes the *gossip on gossip* algorithm to propagate the block graph structure, and achieve the consensus by link analysis in the DAG, this method is proved to be Byzantine fault-tolerant and does not rely on voting. Blockmainia [36] is based on the original PBFT design, but its underlying log is DAG-based. Some of the side chain methods also borrow the idea of DAG, such as nano [37] and VITE [38]. These systems, in reality, rely on centralized methods to maintain their stability.

Social network analysis has widely adopted the method of streaming graph computing [39,40,41], which deals with how to quickly maintain information on a temporally or spatially changing graph without traversing the whole graph. We view the DAG-based method as a streaming graph problem, which is about computing the total order and achieving consensus without consuming more computing power. In distributed database systems, the problem of replicating data across machines is a well-studied topic [42]. Due to the bitcoin network's low efficiency, there are multiple ways to accelerate the message passing efficiency [43]. However, they did not deal with network complexity. We viewed scaling the DAG system in the network of growing size and topological complexity as another challenging issue and proposed our gossip solution. This paper's main contribution is how to utilize the streaming graph analysis methods and new gossip protocol to enable real decentralized, and stabilized growing DAG system.

## 2 Basic design

### 2.1 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 the parent for 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 $\forall b \neq b'$, $\text{Hash}(b) \neq \text{Hash}(b')$.

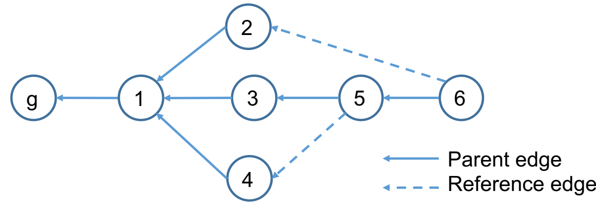


**Fig. 1.** Example of the StreamNet data structure.

### 2.2 StreamNet Architecture

**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   **if** *Generate block* $b$ **then**
8     $a \leftarrow Pivot(G, g)$ ;
9     $r \leftarrow MCMC(G, g)$ ;
10     $G \leftarrow < B \cup b, g, P \cup < b, a >, E \cup < b, a > \cup < b, r >>$ ;
11     Broadcase updated G to neighbors ;
12 **end**

Figure 2 presents the architecture of StreamNet; it is consists of multiple StreamNet machines. Each StreamNet machine will grow its DAG locally and will broadcast the changes using the gossip protocol. Eventually, every machine will have a unified view of DAG. By calling the 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 accepting the transactions from the HTTP API. Moreover, there will be a block generator to pack a certain amount of transactions into a block, and it firstly finds a parent and reference block to attach the new block to, based on the hash information of these two blocks and the metadata of the block itself, it will then perform the proof of work (POW) to calculate the nonce for the new block. Algorithm 1 summarize the server logic for a StreamNet node. In the algorithm, the way to find parent block is by $Pivot(G, g)$. Furthermore, the way to find a reference block is by calling $MCMC(G, g)$, which is the Markov Chain

Monte Carlo (MCMC) random walk algorithm [2]. The two algorithms will be described in the later section.

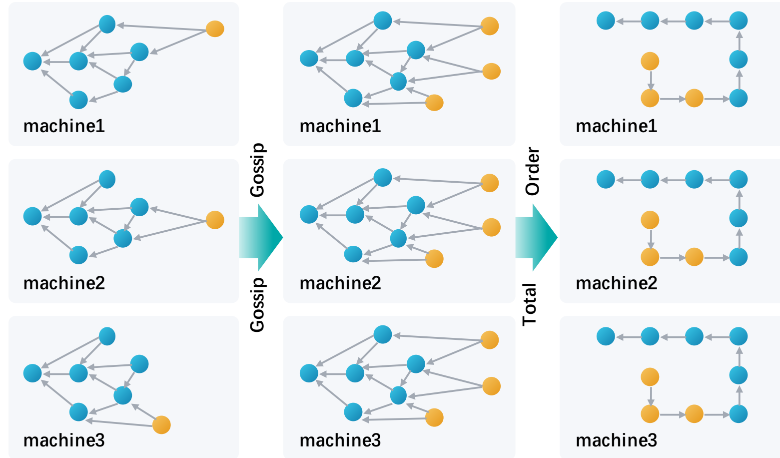


**Fig. 2.** StreamNet architecture.

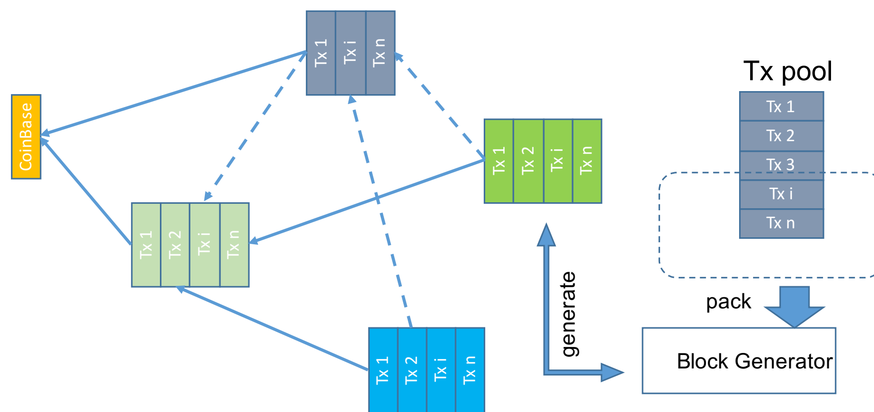


**Fig. 3.** One node in StreamNet protocol.

### 2.3 Consensus protocol

Based on the predefined data structure, to present the StreamNet consensus algorithm, we firstly define several utility functions and notations, which is a variation from the definition in the Conflux paper [1]. Chain() returns the chain from the genesis block to a given block following only parent edges. $\overline{Chain(G,b)}$ returns all blocks except those in the chain. Child() returns the set of child blocks of a given block. Sibling() returns the set of siblings of a given block. Subtree() returns the subtree 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 generated before a given block (but including the block itself). After() returns the set of blocks that are immediately generated after a given block. Later() returns the set of blocks generated after a given block (but including the block itself). SubGraph() returns the subgraph by removing blocks and edges except for the initial set of blocks. ParentScore() presents the weight of blocks, and each block has a score when referenced as a parent. Score() presents the weight of blocks, and each block achieves a score when attaching to the graph. TotalOrder() returns the 'flatten' order inferred from the consensus algorithm. Figure 4 represents the definition of these utility functions.

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

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

$$\overline{Chain(G,b)} = \{b' | b' \in B, b' \notin Chain(G,b)\}$$

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

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

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

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

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

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

$$Later(G,b) = (U_{i \in After(G,b)} Later(G,i)) \cup \{b\}$$

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

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

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

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

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

**Fig. 4.** The Definitions of Chain(), Child(), Sibling(), Subtree(), Before(), Past(), After(), Later(), SubGraph(), ParentScore(), Score(), and TotalOrder().

**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 ← b
2 do
3     for b′ ∈ Child(G, t) do
4         P_bb′ = e^{αScore(G,b′)} / Σ_{z:z→b} e^{αScore(G,z)}
5     end
6     t ← choose b″ by P_bb″
7 while Score(G,t) != 0;
8 return t ;
```

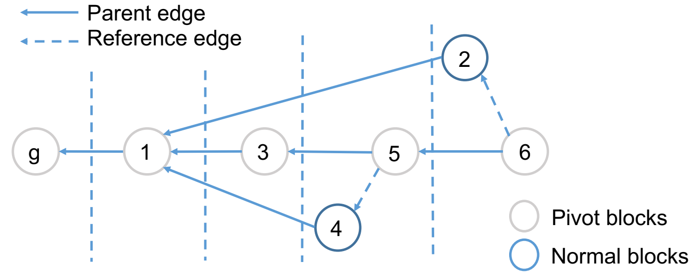


**Fig. 5.** An example of total order calculation.

**Parent tip Selection by pivotal chain** 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 pivoting chain starting from the genesis block $g$. The algorithm recursively advances to the child block, whose corresponding sub-tree has the most significant number of children. Which is calculated by $ParentScore(G, b)$ 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 pivoting chain defines an epoch, 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 pivoting chain is $< g, 1, 3, 5, 6 >$, and the epoch of block 5 contains two blocks 4 and 5.

**Reference tip selection by MCMC** The tip selection method by using Monte Carlo Random Walk (MCMC) is as Algorithm 2 shows. Each random walk step, starting from the genesis, will choose a child to jump to, and the probability of

**Algorithm 3:** PIVOT($G$, $b$).

**Input:** The local state $G = < B, g, P, E >$ and a starting block $b \in B$
**Output:** The tip in the pivot chain

```
1  do
2  |  b′ ← Child(G, b) ;
3  |  tmpMaxScore ← -1 ;
4  |  tmpBlock ← ⊥ ;
5  |  for b′ ∈ Child(G, b) do
6  |  |  pScore ← ParentScore(G, b′) ;
7  |  |  if score > tmpMaxScore || (score = tmpMaxScore
   |  |    and Hash(b′ ) < Hash(tmpBlock) then
8  |  |  |  tmpMaxScore ← pScore ;
9  |  |  |  tmpBlock ← b′ ;
10 |  |  end
11 |  end
12 |  b ← tmpBlock ;
13 while Child(G,b) != 0;
14 return b ;
```

jumping from one block to the next block will be calculated using the formula in the algorithm. $\alpha$ in the formula is a 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 algorithm returns until it finds a tip.

**Total Order** Algorithm 4 defines StreamNetOrder(), which corresponds to our block ordering algorithm. Given the local state $G$ and a block $b$ in the pivoting chain, StreamNetOrder($G$, $b$) returns the ordered list of all blocks that appear in or before the epoch of $b$. Using StreamNetOrder(), the total order of a local state $G$ is defined as TotalOrder($G$). The algorithm recursively orders all blocks in previous epochs(i.e., the epoch of $P(b)$ and before). It then computes all blocks in the epoch of $b$ as $B_{\Delta}$. It topologically sorts all blocks in $B_{\Delta}$ and appends it into the result list. The algorithm utilizes a unique hash to break ties. In Figure 5, the final total order is $< g, 1, 3, 4, 5, 2, 6 >$.

### 2.4 The UTXO model

In StreamNet, the transactions utilize the unspent transaction out (UTXO) model, which is 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 the storage of UTXO in StreamNet and how the conflict is resolved. Two blocks referenced the same block with Alice having five tokens and constructing the new transaction out, representing the transfer of token to Bob and Jack, respec-

**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 $G$

1 $L = \perp$
2 **do**
3 $\quad p \leftarrow$ Parent$(G, b)$ ;
4 $\quad B_\triangle \leftarrow$ Past$(G,b)$ - Past$(G,p)$ ;
5 $\quad$ **do**
6 $\quad\quad G' \leftarrow SubGraph(B_\triangle)$ ;
7 $\quad\quad B'_\triangle \leftarrow$ {x || Before$(G',x) = 0$} ;
8 $\quad\quad$ Sort all blocks in $B'_\triangle$ in order as $b'_1, b'_2, ..., b'_k$
9 $\quad\quad$ such that $\forall 1 \leq i \leq j \leq k$, Hash$(b'_i) \leq$ Hash$(b'_j)$ ;
10 $\quad\quad L \leftarrow L + b'_1 + b'_2 + ... + b'_k$ ;
11 $\quad\quad B_\triangle \leftarrow B_\triangle$ - $B'_\triangle$ ;
12 $\quad$ **while** $B_\triangle \neq 0$;
13 $\quad b = p$ ;
14 **while** $b$ != $g$;
15 **return** $L$ ;

tively. However, after calling $totalOrder()$, the Bob transfer block precedes the Jack transfer block; thus, the next block will be discarded.

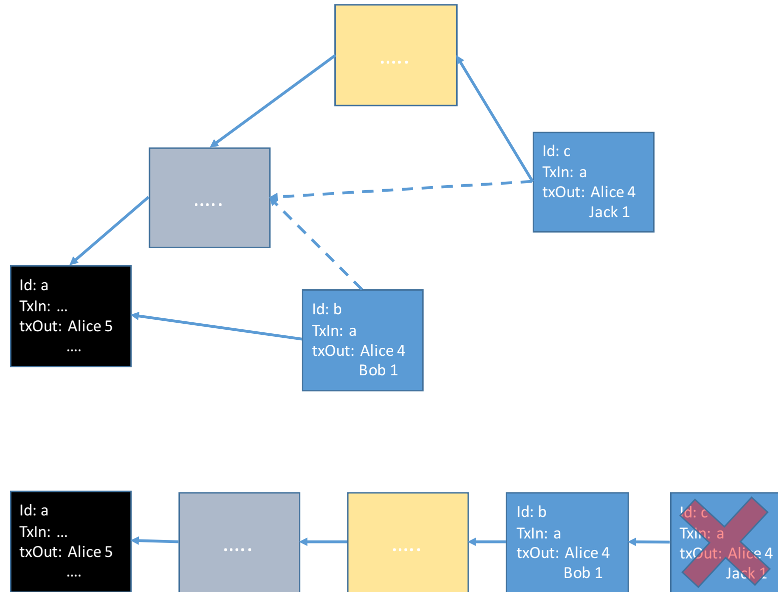


**Fig. 6.** An example of UTXO.

### 2.5 Gossip Network

In the bitcoin and IOTA network, the block information is disseminated in a direct mail way [42]. Suppose there are $N$ nodes and $L$ links in the network, for a block of size $B$, to spread the information of it, the direct mail algorithm will have a total complexity of $O(LB)$. Moreover, the average complexity for a node will be $O(\frac{LB}{N})$ In the chain based system, and this is fine because the design of the system already assumes that the transaction rate will below. However, in the DAG-based system, this type of gossip manner will result in low scalability due to the high throughput of the block generation rate and will result in network flooding. What is worse, consider the heterogeneously and long diameters of network topology, the convergence of DAG will take a long time, which will cause the delay of confirmation time of blocks.

### 2.6 Differences with other DAG protocols

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

**IOTA** The major difference with IOTA is in three points:

- Firstly, the IOTA tip selection algorithm's two tips are all randomly chosen, and ours is one deterministic which is for the total ordering purposes and one by random which is for maintaining the DAG property;
- Secondly, the IOTA consensus algorithm is not purely decentralized, it relies on a central coordinator to issue milestones for multiple purposes, and our algorithm does not depend on such a facility.
- Lastly, in IOTA, there is no concept of total order, and there are three 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 this tip references a block, its credibility is increased by 1. After $N$ selections have been cited $M$ times, then the credibility is $M/N$.

**Conflux** The major difference with Conflux is in two points:

- Firstly, Conflux will approve all tips in the DAG along with the parent, which is much more complicated than our MCMC based two tip method. Moreover, when the width of DAG is high, there will be much more space needed to maintain such data structure.

– Secondly, the 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 contributions to our streaming graph-based optimizations, which will be discussed in the next chapter. In Conflux paper, there is no description of how to deal with the complexity paired with the growing graph.

### 2.7 Correctness

**Safety & Liveness** Because StreamNet utilizes the GHOST rule to select the pivoting chain, which is the same as in Conflux. Thus, it shares the same safety and correctness property as Conflux. Although the choice of reference chain in StreamNet is different from Conflux, it only affects the inclusion rate, which is the probability of a block to be included in the total order.

**Confirmation** According to Theorem 10 in [30] and the deduction in [1], given a period of $[t-d,t]$, and block $b$ in pivot chain in this period, the chance of $b$ kicked out by its sibling $b'$ is no more than $Pr(b_{drop})$ in (1). Which is the same as in Conflux.

$$Pr(b_{drop}) \leq \sum_{k=0}^{n-m} \zeta_k q^{n-m-k+1} + \sum_{k=n-m+1}^{\infty} \zeta_k \zeta_k = e^{-q\lambda_h t \frac{(-q\lambda_h t)^k}{k!}} \tag{1}$$

Followed by the definitions in Conflux paper [1], in (1), $n$ is the number of blocks in the subtree before $t$, $m$ is the number of blocks in subtree of $b'$ before $t$. $\lambda_h$ is an honest node's block generation rate. $q(0 \leq q \leq 1)$ is the attacker's block generation ratio with respect to $\lambda_h$. From the equation, we can conclude that with the time $t$ goes, the chance of a block $b$ in the pivoting chain to be reverted is decreased exponentially.

## 3 Optimization Methods

One of the biggest challenges to maintain the stability of the DAG system is that, as the local data structure grows, the graph algorithms ($Pivot()$, $MCMC()$, $StreamNetOrder()$), relies on some of the graph operators that need to be recalculated for every newly generated block, which is very expensive. Table 1 list all the expensive graph operators that are called. Suppose the depth of the pivoting chain is $d$, then we give the analysis of complexity in the following way. $ParentScore()$ and $Score()$ rely on the breadth-first search ($BFS$), and the average $BFS$ complexity would be $O(|B|)$, and for each MCMC() and Pivot() called the complexity would be in total $O(|B|^2)$ 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 is the classical topological sorting problem, and the complexity in the StreamNetOrder() would be $O(|B|)$.

**Table 1.** Analysis of Graph properties calculation

| Graph Property | Algorithm used | Complexity | Tot |
|---|---|---|---|
| ParentScore(G, b) | Pivot() | $O(\|B\|)$ | $O(\|B\|^2)$ |
| Score(G, b) | MCMC() | $O(\|B\|)$ | $O(\|B\|^2)$ |
| Past(G,b) - Past(G,p) | StreamNetOrder() | $O(\|B\|)$ | $O(\|B\|*d)$ |
| TopOrder(G, b) | StreamNetOrder() | $O(\|B\|)$ | $O(\|B\|)$ |

Considering new blocks are generated 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 amortized 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.

### 3.1 Optimization of Score() and ParentScore()

**Algorithm 5:** UPDATESCORE($G$, $b$).

**Input:** Graph $G$, Block $b$, Score map $S$
**Output:** Updated score map $S$

```
1  Q = [b] ;
2  visited = {} ;
3  while Q != Ø do
4  |   b' = Q.pop() ;
5  |   for b'' ∈ Before(G, b') do
6  |   |   if b'' ∉ visited ∧ b'' != ⊥ then
7  |   |   |   Q.append(b'') ;
8  |   |   |   visited.add(b'') ;
9  |   end
10 |   S[b'] + + ;
11 end
12 return S ;
```

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 block's scores in the map that are referenced by the new block. The skeleton of the UpdateScore() algorithm is as Algorithm 5 shows.

**Algorithm 6:** GetDiffSet($G$, $b$, $C$).

```
    Input:  Graph G, Block b, covered block set C
    Output: diff set D ← Past(G,b) − Past(G,p)
 1  D = ∅ ;
 2  Q ← [b] ;
 3  visited = {b} ;
 4  p = Parent(G,b) ;
 5  while Q != ∅ do
 6  |   b' = Q.pop() ;
 7  |   for b'' ∈ Before(G,b') do
 8  |   |   if IsCovered(G,p,b'',C) ∧ b'' != ⊥ then
 9  |   |   |   Q.append(b'') ;
10  |   |   |   visited.add(b'') ;
11  |   end
12  |   D.add(b') ;
13  |   C.add(b') ;
14  end
15  return D ;
```

**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 ← [b'] ;
 2  visited = {b} ;
 3  while Q != ∅ do
 4  |   b'' = Q.pop() ;
 5  |   for t ∈ Child(G,b'') do
 6  |   |   if t = p then
 7  |   |   |   return true ;
 8  |   |   else if t ∉ visited ∧ t ∉ C then
 9  |   |   |   Q.add(t) ;
10  |   |   |   visited.add(t) ;
11  |   end
12  end
13  return false ;
```

### 3.2 Optimization of Past(G,b) - Past(G,p)

We abbreviate the Past(G,b) - Past(G,p) to calculate $B\delta$ 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 all 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 $p$. If yes, $b'$ would not be added to the forward $BFS$ queue. To avoid the complexity of the backward $BFS$, we add the previously calculated diff set 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 algorithm 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 performs 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 has three children. If it detects 3, which is the parent of 5, it will stop searching promptly. If it continues searching on 2 or 4, these two blocks would not be added to the search queue, because they are already in the covered set.

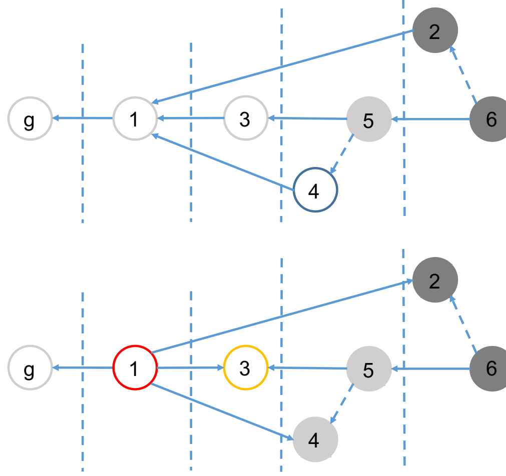


**Fig. 7.** Example of the streaming get diff set method.

### 3.3 Optimization of TopOrder()

The topological order is used in sorting the blocks in the same epoch. To get the topological order, every time, there needs a top sort of the whole DAG from 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; its topological position will be as (1) shows. This step can be done in $O(1)$

$$TopScore(G, b) \leftarrow min(TopScore(G, Parent(b)), TopScore(G, Reference(b)))+1 \quad (2)$$

To summarize, the optimized streaming operators can achieve performance improvement as Table 2 shows.

**Table 2.** Analysis of Graph properties calculation

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

### 3.4 Genesis Forwarding

The above algorithm solved the problem of how to dynamically maintaining the information needed for graph computation. However, it still needs to update the information until the genesis block. With the size of the graph growing, the updating process will become harder to compute. With the growth of DAG size, the old historical confirmed blocks are being confirmed by more and more blocks, which are hard to be mutated. Furthermore, the exact probability can be computed in formula (1). Hence, we can design a strategy to forward the genesis periodically and fix the historical blocks into a total ordered chain. The criteria to forward the genesis are based on the threshold of ParentScore(). Suppose we define this threshold as $h = n - m$, then we only forward the genesis if:

$$\exists b | b \in Chain(G, g), for \forall b' | b' \in \overline{Chain(G, g)}, such that ParentScore(b) > ParentScore(b')+h \quad (3)$$

In Figure 8, we set $h = 5$, and there are three side chains with $\forall b' | b' \in \overline{Chain(G, g)}$, $ParentScore(b') <= 4$. And in pivot chain, there are multiple blocks $b$ that has $ParentScore(b) >= 9$, they are candidates for the new genesis, we choose the block with minimum $ParentScore$ as the new genesis.

Besides, after the new genesis has been chosen, we will induce a new DAG in memory from this genesis; furthermore, persist the 'snapshot' total order (Conflux paper has the same definition, but it does not show the technical detail, we do not view it trivial) in the local database. Once the total order is queried, a total order based on the current DAG will be appended to the end of the

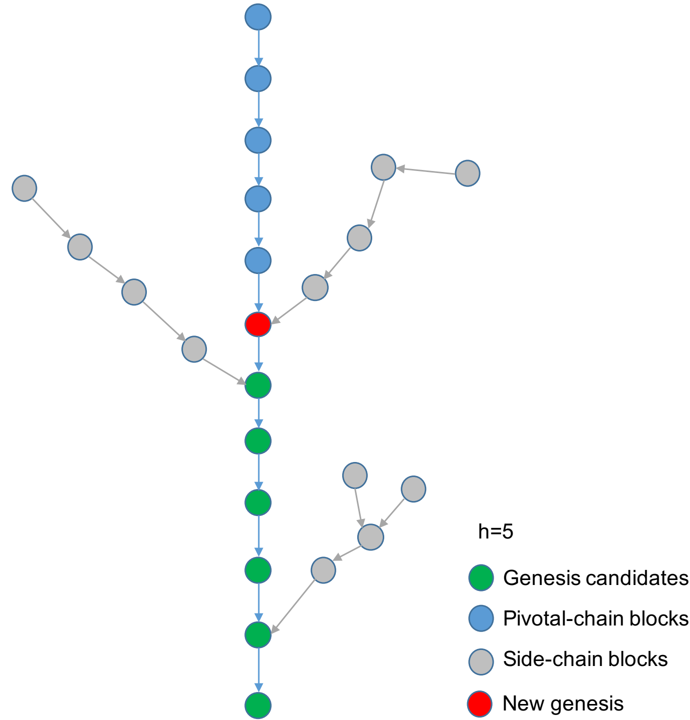


**Fig. 8.** Example of genesis forward method.

historical snapshot total order and be returned. Also, the vertices in the UTXO graph that belongs to the fixed blocks will be eliminated from the memory and be persisted to disk as well. The algorithm is as Algorithm 8 shows.

### 3.5 The Direct Signal Gossip Protocol

There are solutions in [42] to minimize the message passing in the gossip network. Moreover, in Hyperledger [44] they have adopted the PUSH and PULL model for the gossip message propagation. However, their system is aiming at permissioned chain. Suppose the size of the hash of a block is $H$, we designed the direct signal algorithm. The algorithm is divided into two steps, once a node generates or receives a block, it firstly broadcast the hash of the block, this is the PUSH step. Once a node receives a hash or a set of a hash, it will pick one source of the hash for the block content, and this is the PULL step. The direct signal algorithm's complexity will be $O(LH + NB)$ and for a node averaged to $O(\frac{LH}{N} + 1)$ The algorithm is as Algorithm 9 shows.

**Algorithm 8:** Genesis Forward Algorithm.

```
    Input: Graph G =< B, g, P, E >
 1  while Node is running do
 2  |   if ∃b satisties (3) then
 3  |   |   O = TopOrder(G, g);
 4  |   |   g′ ← b ;
 5  |   |   G′ ← induceGraph(G, g′) ;
 6  |   |   pS = ParentScore(G′, g′);
 7  |   |   S = Score(G′, g′);
 8  |   |   O′ = TopOrder(G′, g′);
 9  |   |   G ← G′ ;
10  |   |   persist O − O′ ;
11  |   sleep (t) ;
12  end
```

**Algorithm 9:** The Direct Signal Gossip Algorithm.

```
    Input: Graph G =< B, g, P, E >
 1  while Node is running do
 2  |   if Generate block b then
 3  |   |   Broadcast b to neighbors ;
 4  |   if Receive block b then
 5  |   |   h ← Hash(b) ;
 6  |   |   cache[h] ← b ;
 7  |   |   Broadcast h to neighbors ;
 8  |   if Received request h from neighbor n then
 9  |   |   b ← cache[h] ;
10  |   |   Send b to n ;
11  |   if Received hash h from neighbor n then
12  |   |   b ← cache[h] ;
13  |   |   if  b = NULL then
14  |   |   |   Send request h to n ;
15  end
```

# 4 Experimental Results

## 4.1 Implementation

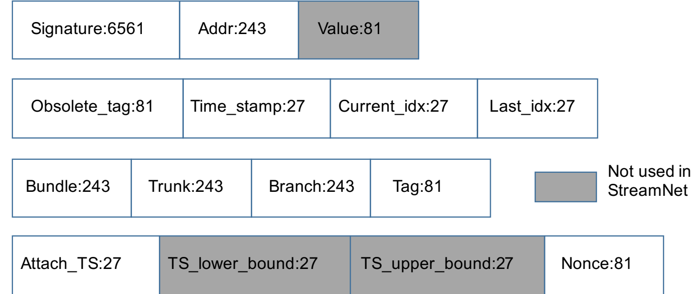


**Fig. 9.** Block header format, the main transaction information is stored in the signature part. The addr is sender's address, the timestamp is the time the block has been created, current/last index and the bundle is used for storing the bundle information, trunk and branch are the hash address to store the parent and reference location, the tag is used for store some tagging information, addtach_TS is when the block is attached to the StreamNet, the nonce is used in POW calculation.

We have implemented the StreamNet based on the IOTA JAVA reference code (IRI) v1.5.5 [45]. We forked the code and made our implementation; the code is freely available at [46]. In this paper, we use version v0.1.4-streamnet in the v0.1-streamnet beta branch.

- The features we have adopted from the IRI are:
  - The block header format, as shown in Figure 9. Some of the data segments are not used in StreamNet, which are marked grey.
  - Gossip network, the network is a bi-directional network in which every node will send and receive data from its peers;
  - Transaction 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 including the ledger validation part;
  - The milestone issued by coordinators, which is a centralized setup.
- 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, and we have applied the streaming method directly in the algorithms;
  - The UTXO logic stored in the signature part of the block header used the graph data structure to store UTXO as well.
  - In IOTA's implementation, the blocks are stored in the RocksDB [47] as the persistence layer, which makes it inefficient to infer the relationships between blocks and calculate graph features. In our implementation, we introduced an in-memory layer to store the relationships between blocks, such that the tip selection and total ordering algorithm will be accelerated.

### 4.2 Environment Set Up

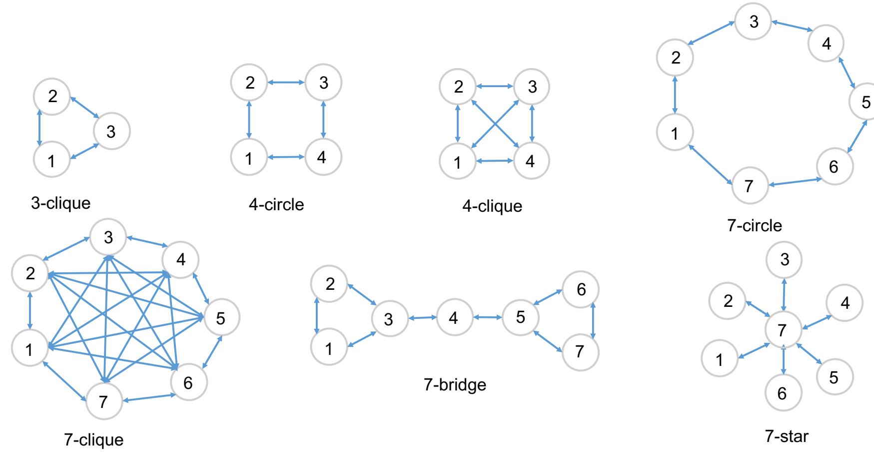


**Fig. 10.** Cluster set up for different network topologies.

We have used the AWS cloud services with 7 virtual machines, for each node, it includes a four-core AMD EPYC 7571, with 16 Gb of memory size and 296Gb 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 topologies set up of nodes, which are shown in Figure 10, these configurations are aiming to test:

- The performance when the cluster connectivity is high (congestion of communications, like 3-clique, 4-clique, 7-clique, and 7-star);
- The performance when the cluster diameter is high (long hops to pass the message, like 4-circle, 7-circle, 7-bridge);

As for the data, we have created 1,000 accounts, with the genesis account having 1,000,000,000 tokens in the coinbase block. We divided the accounts into two groups (each group will have 500 accounts), the first group will participate in the ramp-up step, which means the genesis account will distribute the tokens to these accounts. Moreover, for comparison, we have issued four sets of different size transactions (5000, 10000, 15000, and 20000), respectively. In the execution step, the first group of accounts will issue transactions to the second group of accounts, which constructs a bipartite spending graph. Since there are more transactions than the number of accounts, there will be double-spend manners in this step. The number of threads in this procedure is equal to the number of nodes for each configuration. Jmeter [48] is utilized as the driver to issue the transactions, and Nginx [49] is used to evenly and randomly distribute the requests to different nodes.

### 4.3 Results and Discussions

**Block generation rate test** To test the block generation rate, we set each block in StreamNet to have only one transaction. Furthermore, the performance on this configuration is as Figure 11 shows. First, as the size of the cluster grows, the network will witness little performance loss on all of the data scales. In the experiment, we can also see that with the growth of the data, the average TPS on most of the configurations have grown a little bit (some outliers need our time to triage), this is because the genesis forwarding algorithm needs some ramp-up time to get to the stable growth stage. Considering the system is dealing with a growing graph instead of a chain and the complexity analysis in the previous section, the experiment clearly shows that our streaming algorithm sheds light on how to deal with the growing DAG.

**Bundle transaction test** By default, each block in StreamNet will support bundle transactions. We set each bundle to contain 20 transactions, and for each block, there are approximately 3 transactions included. The performance on this configuration is as Figure 12 shows. In this experiment, we can see that the performance (TPS) comparing with the block test improved more than twice. This is because there will be less POW works to be done. Besides, with the growth of the data, we do not witness a noticeable performance downturn. Nevertheless, there are some performance thrashing in the experiment, which needs more study.

## 5 Conclusion

In this paper, we proposed a way to compute how to grow the blocks in the growing DAG based blockchain systems. And how to maintain the total order as the DAG structure is dynamically turning larger. We referred one of the earliest DAG implementation IRI to conduct our own experiments on clusters of different size and topology. Despite the network inefficiency in the IRI implementation,

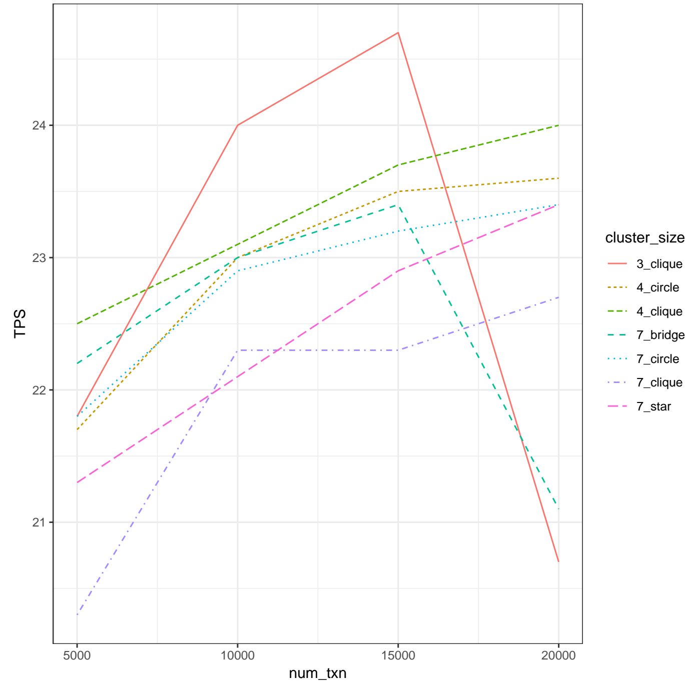


**Fig. 11.** Experimental results for block generation rate.

our method is proven to be able to tolerate the increasing complexity of the graph computation problems involved. This is due to the streaming graph computing techniques we have introduced in this paper.

## References


1. Chenxing Li, Peilun Li, Wei Xu, Fan Long, and Andrew Chi-chih Yao. Scaling nakamoto consensus to thousands of transactions per second. *arXiv preprint arXiv:1805.03870*, 2018.
2. Serguei Popov. The tangle. *cit. on*, page 131, 2016.
3. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
4. JW MICHAEL, ALAN COHN, and JARED R BUTCHER. Blockchain technology. *The Journal*, 2018.
5. Alex Tapscott and Don Tapscott. How blockchain is changing finance. *Harvard Business Review*, 1, 2017.
6. Kari Korpela, Jukka Hallikas, and Tomi Dahlberg. Digital supply chain transformation toward blockchain integration. In *proceedings of the 50th Hawaii international conference on system sciences*, 2017.
7. Feng 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*, pages 1–6. IEEE, 2016.

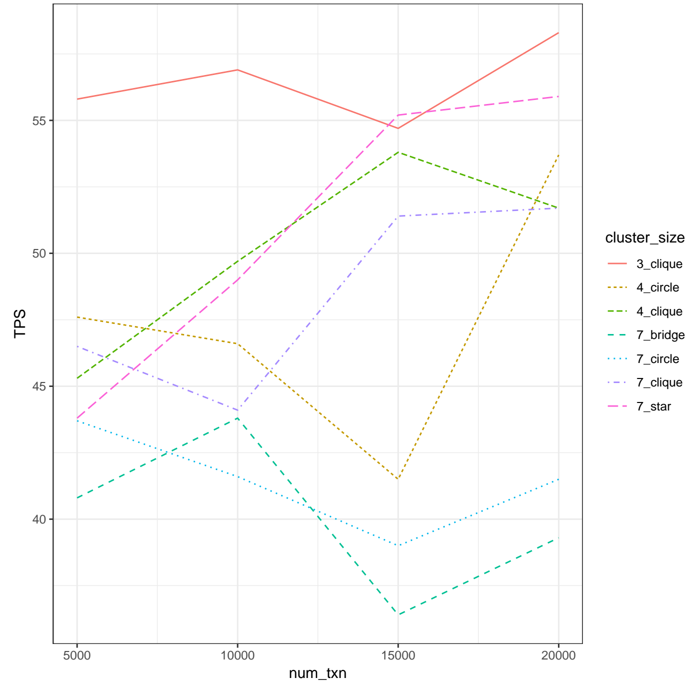


**Fig. 12.** Experimental results for bundle transaction.


8. Saveen A Abeyratne and Radmehr P Monfared. Blockchain ready manufacturing supply chain using distributed ledger. 2016.
9. Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. Medrec: Using blockchain for medical data access and permission management. In *Open and Big Data (OBD), International Conference on*, pages 25–30. IEEE, 2016.
10. Xiao Yue, Huiju Wang, Dawei Jin, Mingqiang Li, and Wei Jiang. Healthcare data gateways: found healthcare intelligence on blockchain with novel privacy risk control. *Journal of medical systems*, 40(10):218, 2016.
11. Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *Ieee Access*, 4:2292–2303, 2016.
12. Eric Lombrozo, Johnson Lau, and P WUILLE. Segregated witness, 2015.
13. Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.
14. Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, pages 1–47, 2017.
15. Evan Duffield and Daniel Diaz. Dash: A payments-focused cryptocurrency, 2018.
16. TRON Foundation. Advanced decentralized blockchain platform. *Whitepaper) https://tron.network/static/doc/white_paper_v_2_0.pdf*, 2018.
17. Bernardo Machado David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. *IACR Cryptology ePrint Archive*, 2017:573, 2017.

18. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.
19. LM Goodman. Tezosa self-amending crypto-ledger white paper. *URL: https://www. tezos. com/static/papers/white_paper. pdf*, 2014.
20. Spoke Matthew and Engineering Team Nuco. Aion: Enabling the decentralized internet. *Aion project yellow paper*, 151:1–22, 2017.
21. Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.
22. Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *NSDI*, pages 45–59, 2016.
23. Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42. ACM, 2016.
24. Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296, 2016.
25. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.
26. David Schwartz, Noah Youngs, Arthur Britto, et al. The ripple protocol consensus algorithm. *Ripple Labs Inc White Paper*, 5, 2014.
27. David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation*, 2015.
28. Jae Kwon and Ethan Buchman. Cosmos: A network of distributed ledgers. *URL https://cosmos. network/whitepaper*, 2016.
29. Yonatan Sompolinsky and Aviv Zohar. Accelerating bitcoins transaction processing. *Fast Money Grows on Trees, Not Chains*, 2013.
30. Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.
31. Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 528–547. Springer, 2015.
32. Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: Serialization of proof-of-work events: confirming transactions via recursive elections, 2016.
33. Yonatan Sompolinsky and Aviv Zohar. Phantom, ghostdag.
34. Anton Churyumov. Byteball: A decentralized system for storage and transfer of value. *URL https://byteball. org/Byteball. pdf*, 2016.
35. Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep.*, 2016.
36. George Danezis and David Hrycyszyn. Blockmania: from block dags to consensus. *arXiv preprint arXiv:1809.01620*, 2018.
37. Colin LeMahieu. Nano: A feeless distributed cryptocurrency network. *Nano [Online resource]. URL: https://nano. org/en/whitepaper (date of access: 24.03. 2018)*, 2018.
38. Chunming Liu, Daniel Wang, and Ming Wu. Vite: A high performance asynchronous decentralized application platform.
39. David Ediger, Jason Riedy, David A Bader, and Henning Meyerhenke. Tracking structure of streaming social networks. In *2011 IEEE International Parallel &*

*Distributed Processing Symposium Workshops and PhD Forum*, pages 1691–1699. IEEE, 2011.
40. Oded Green, Robert McColl, and DA Bader. A fast algorithm for incremental betweenness centrality. In *Proceeding of SE/IEEE international conference on social computing (SocialCom)*, pages 3–5, 2012.
41. David Ediger, Rob McColl, Jason Riedy, and David A Bader. Stinger: High performance data structure for streaming graphs. In *2012 IEEE Conference on High Performance Extreme Computing*, pages 1–5. IEEE, 2012.
42. Alan Demers, Dan Greene, Carl Houser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. *ACM SIGOPS Operating Systems Review*, 22(1):8–32, 1988.
43. Uri Klarman, Soumya Basu, Aleksandar Kuzmanovic, and Emin Gün Sirer. bloxroute: A scalable trustless blockchain distribution network whitepaper.
44. Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, page 30. ACM, 2018.
45. Iota reference implementation. *https://github.com/iotaledger/iri.*
46. Streamnet reference implementation. *https://github.com/triasteam/iri.*
47. Rocksdb reference implementation. *http://rocksdb.org.*
48. Emily H Halili. *Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites.* Packt Publishing Ltd, 2008.
49. Clément Nedelcu. *Nginx HTTP Server: Adopt Nginx for Your Web Applications to Make the Most of Your Infrastructure and Serve Pages Faster Than Ever.* Packt Publishing Ltd, 2010.