Flink: The Fast Distributed Payment Network

M. Grasic*

info@flinkcoin.org

Abstract

Cryptocurrency appeared as an interesting technology that promised to revolutionize the payment system. But over time, cryptocurrencies have been backed into a corner and confined to store of value speculation. Cryptocurrencies are almost never used as a means of payment and most transactions only take place through centralized exchange systems. Here we introduce a feeless cryptocurrency based on a block-lattice design with asynchronous transactions and block confirmations. Flink is designed for high transaction speeds of more than 4000 transactions per second with low latency of around 3 seconds or less. With these features, Flink is the ideal cryptocurrency for consumer transactions as well as high-frequency and zero-fee business transactions. In terms of security, all cryptographic elements are integrated into a replaceable design so that cryptographic functions can be swapped out in case of quantum security requirements.

*URL: `https://www.flinkcoin.org/`

I. INTRODUCTION

Distributed ledger technology (DLT) (Scardovi [1], Maull *et al.* [2]) has been on the rise since the introduction of Bitcoin in Nakamoto [3]. Since Bitcoin, many new and promising DLT projects have been introduced, but none have truly achieved global adaptation as a payment system. DLT-based currencies or cryptocurrencies have mostly been isolated to act as stores of value, much like gold, and the idea of a global, fair and cheap payment system has mostly been abandoned. This is due to government regulations, but also high transaction fees and slow transaction verifications. For a DLT currency to be truly effective and better than government-backed currencies, transaction speeds must be near-instantaneous with zero or negligible fees. Without this improvement, DLT currencies offer no real value to the public or consumers, and thus no reason to use them.

A. Motivation

Although many new DTL technologies have been introduced since the launch of Bitcoin. The actual use of a cryptocurrency as a payment system is still lacking as more and more projects gain traction as a DeFI (decentralised finance) use case. Therefore, there is still a lot of room for improvement.

The perfect payment system is:

- Feeless or almost zero fee: Zero fee makes sending money attractive and enables various business opportunities.
- Fast: Low latency is important to enable fast payments for goods.
- Inexpensive to operate: The system must be available and easy to operate.
- Secure: Security is of course of paramount importance, as trust in the system is required.

Nano (LeMahieu [4]) was one of the first DAG (Directed Acyclic Graph) (Devarajan and Karabulut [5]) based cryptocurrencies that promised all these features. DAG is now implemented in many projects alike (Popov [6], Churyumov [7]).The DAG-based cryptocurrencies have a performance advantage in that transactions can be confirmed asynchronously, rather

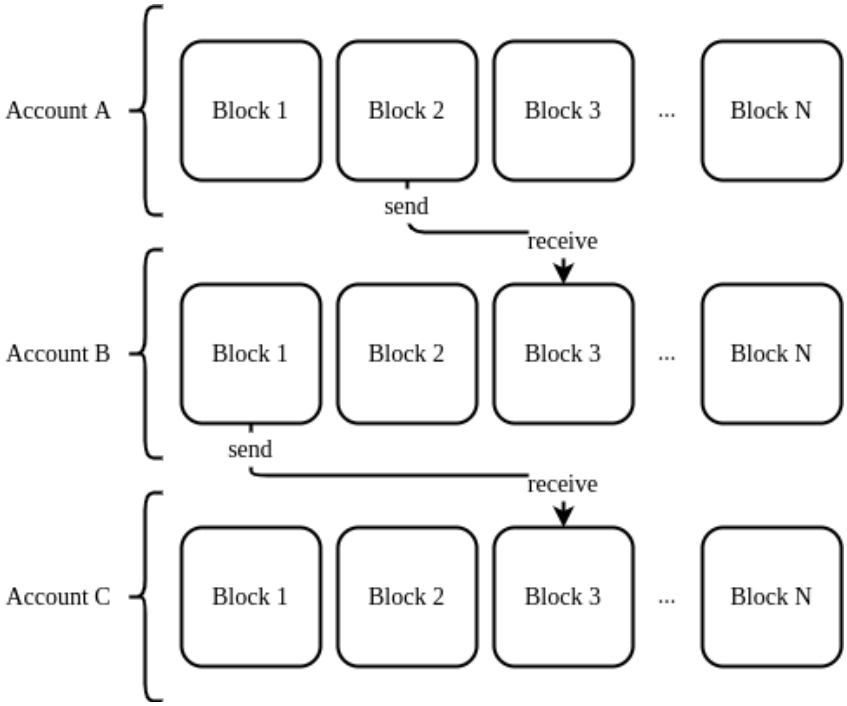

Figure 1: Block-lattice data structure.

than synchronously as in a linear system like the blockchain. This drastically lowers transaction confirmation times, making them mostly instantaneous (in the range of 1-3s) and also improves transaction throughput to over 1000 tps. Feeless DAG systems achieve most of these goals, but availability and stability suffer from such a design. Without transaction fees, DDOS attacks are very cheap, and recent attacks on the Nano network have further illustrated this vulnerability.

II. BASIC CONCEPT

Here we introduce Flink, a zero fee, high speed and secure network that is optimized for enterprise use, but can also be run by individuals on commodity hardware. The system is optimized for fast transactions and is based on a peering algorithm that connects the most influential nodes in a way that minimizes block confirmation latency. The system follows a similar architecture to Nano, which is based on a block lattice. Each account has its own blockchain, which is shown on Figure 1. Here, the account chain can only be updated by the account holder. Since the blockchain of the accounts can be changed independently, we do not need a common block to fill up with transactions and the confirmation latency is

greatly reduced.

A. Account Based

Instead of a UTXO transaction design, as used in Bitcoin and many other cryptocurrencies, an account-based design was chosen instead. The advantage of such an approach is simplicity, speed, and usually lower memory requirements. The only disadvantage of this approach is anonymity, which is more difficult to ensure in a DLT design. For this 2nd layer, solutions are planned.

B. Sending Funds

Because each account blockchain can only be changed by the account holder. A money transfer can only be done with two transactions. First, account A performs a send transaction to account B, then account B requests the transaction by a new receive transaction. This approach is more complicated compared to blockchains like Ethereum and Bitcoin, but the advantages lie in the confirmation process, which can be asynchronous and therefore very scalable.

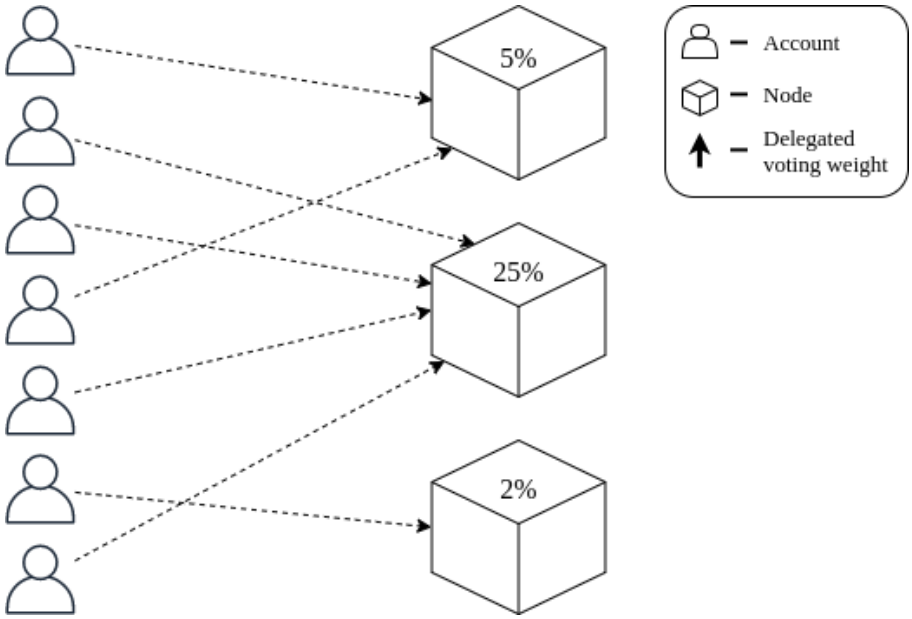

Figure 2: Node weight assignment from accounts.

III. SYSTEM DESIGN

The system is based on a block-lattice-like structure, where each account has its own blockchain. It uses an optimized consensus voting based algorithm that reduces latency and network traffic. The block is optimized for size and speed and represents a user transaction. The node network is also highly interconnected to further reduce transaction latency.

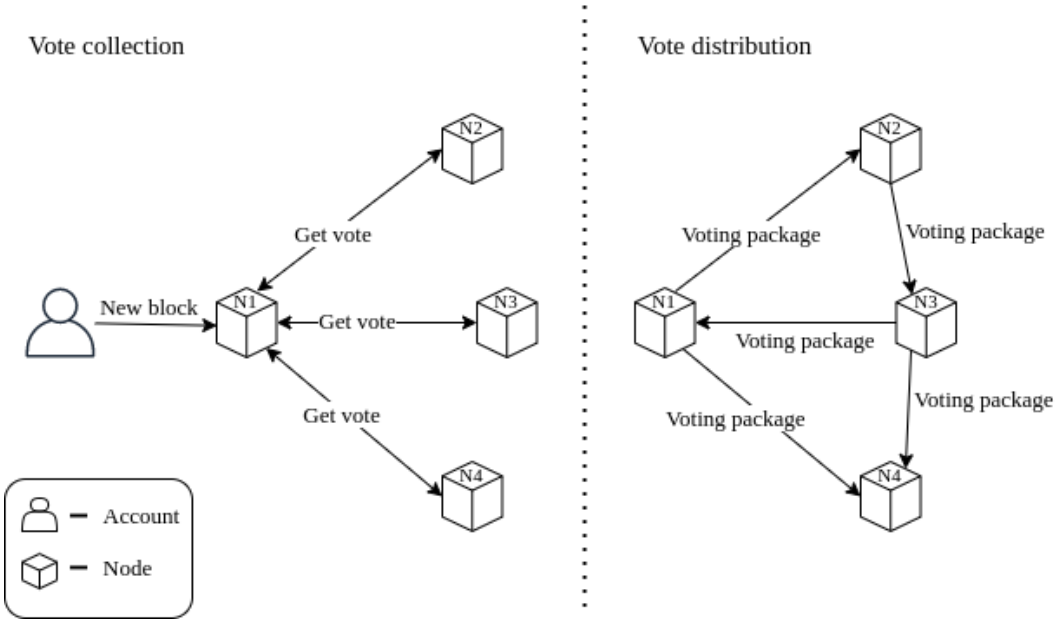

Figure 3: New block voting process.

A. Consensus Algorithm

Consensus is achieved using a voting-based consenous algorithm and recently PoV (Proof of Vote) (Li *et al.* [8]) consensus algorithms become well known in the scientific literature. Like PoW, they are Byzantine fault-tolerant, secure and stable (Zhang [9]). As in ORV (Open Representative Voting) (LeMahieu [4]), voting weights are determined by the account holder's selection of a representative. The representative is assigned a voting weight relative to the balance of all accounts assigned to him, as shown in the Figure 2. This approach is particularly suitable for feeless systems, since the cost of validating a transaction is low and without risk. With PoW (Proof of Work) the cost is very high and with PoS (Proof of Stake) there is potential risk when staking the currency.

Flooding the network directly with votes would lead to message overload and would be

slow and inefficient. To limit the number of votes and messages, some systems choose a middleman who then performs the validation process. While this increases performance, it decreases resilience to a potential attack. Here, we describe a consensus process in which the system is not flooded with votes, but the node receiving the block is tasked with procuring all the necessary vote weights to satisfy the $\left[\frac{W_T}{2} + 1\right] < W_c$ condition, where W_T is the total weight of all the coins and W_C is the weight of the collected votes. First, the node receiving the new block collects the votes of the nodes one by one so that the majority rule is satisfied. Once the votes are collected and the majority rule is satisfied, a voting package is created with the new block and all the votes. This is then sent to all nodes using the network flooding procedure described below, shown on Figure 3.

1. Proof of Vote on Blocks

The user creates a block and sends it to a node. The node that receives the block first is responsible for obtaining the required voting majority before it creates the voting package and injects it into the network. The nodes validate the block and vote independently. The majority weight is calculated with:

$$W_C^X = \sum_{i=1}^{N} W_i^X \tag{1}$$

Where N is the number of votes collected and X is the block voted on. If the majority condition is not met, N can be increased, which in practice means that a vote is requested from more nodes. If all or most nodes are requested and no transaction vote majority is reached, the block is discarded.

B. Network Flooding

In network flooding, a message received by a node is sent over all outgoing connections except the one over which it was received (Ashikur *et al.* [10]). In uncontrolled flooding, each node unconditionally distributes messages to each of its neighbors. Without conditional logic to prevent infinite repetition of the same packet. In controlled flooding, two algorithms are common: SNCF (Sequence Number Controlled Flooding) and RPF (reverse-path forwarding). In SNCF, the node appends its own address and sequence number to the packet

because each node has a memory of addresses and sequence numbers. When it receives a packet in memory, it immediately discards it, while in RPF, the node only forwards the packet.

The system uses a custom SNCF algorithm where each message is given a unique identifier and the nodes remember all the identifiers already processed. The node information is not added to the original message as this would slow down the process, alter the source message and increase the size of the message. Nodes also do not send messages to all connections, but only to connections that are designated for flooding. This limits network traffic while providing multiple direct connections during the voice collection phase. The number of messages needed for a block confirmation can be calculated with:

$$M = N * c_f + N_W * 2 \tag{2}$$

Where M is the number of messages N is the total number of nodes in the network, c_f is the number of connections each node provides for flooding and N_w is the number of nodes required to achieve majority voting.

C. Partition Tolerance

If the network is split into two completly separate partitions α and β, where $\alpha \cap \beta = \oslash$. As long one partition α, β satisfies the condition $\left[\frac{W_T}{2} + 1\right] < W_c$, consenus can be achieved and new blocks can be confirmed (Guo *et al.* [11]). This is possible because W_T is global and does not change when the size of the network nodes is smaller. If the majority condition is not satisfied, the network becomes temporarily unavailable. Here, according to Brewer's CAP theorem in Diack *et al.* [12], availability is sacrificed for consistency and partition tolerance.

Table I: Block with all data definitions.

Field	Description
version	block version, for future changes
timestamp	transaction timestamp, special importance in the system
blockType	block type
previousBlockHash	previous block hash value
accountId	block for which account
representativeNodeId	account chosen representative
balance	current account balance
amount	transaction amount
sendAccountId	receiving account id
receiveBlockHash	block we receive funds from
referenceCode	payment reference code useful for accounting
publicKeys	last valid public keys

D. Block Design

To simplify the design shown in Table I, a common state block was chosen. Having all state information in each block speeds up transaction times and simplifies the design. It also improves node synchronization and the transaction validation process. Although this increases the memory requirements, this is optimized by state pruning.

The design of the system and block is strongly focused on change flexibility. Once a distributed system is in production, it is extremely difficult to change the block design. To ease the burden, we have incorporated a version code so that the block can change from one version to another. Also, the serialization process is done in such a way that new fields can be added easily.

E. Transaction Types

Although the block structure is the same for each transaction. The system still functionally distinguishes between different transactions sent. Flink supports 4 different transaction types: create, send, receive, and update.

1. Create Transaction

Before an account can be used to send or receive, it must first be opened. The account is set up with a zero balance. Also, at the beginning, the first public keys are injected and the representative is selected.

2. Receive Transaction

Before transactions can be received, the account must be created. Once it is successfully created, transactions can be received. The transaction is received by a corresponding block hash (receiveBlockHash) and sent to this account. Therefore, a block hash must be referenced from which we receive the amount. The same block can only be referenced once. The received block updates the balance status.

3. Send Transaction

Before sending a transaction for the first time, the account must have a non-zero balance. Once a non-zero value is detected, the money can be sent to an account (sendAccountId). The account balance is updated with the send and receive block. The balance should never be less than zero.

4. Update Transaction

An update transaction is usually used to assign the representative (representativeNodeId). This can also be done for create, update and receive transactions. But this type of transaction is for when nothing is to be sent or received.

F. Multisig Support

Multisig support is an extremely important feature of any large blockchain. Exchanges and enterprise users will need this in the future to increase security and distribute fund management responsibilities. However, there are also multisig-like approaches where multiple private keys can share a public key. These approaches are quite complicated to integrate

into production environments. Therefore, Flink inherently supports multisig with up to five signatures as a block design for specifying multiple public keys and signatures.

G. Fast Transaction Times

In order for a block to be confirmed, a quorum of votes must be reached (majority vote weight). In order for this to happen quickly, all representatives with high vote weights must be connected to each other so that messaging and confirmation occur as quickly as possible. Each node therefore tries to connect to as many representative nodes as possible. The nodes are also checked for latency so that the fast nodes are connected first.

H. Time-Restricted Transactions

Each block has a creation timestamp. Each transaction can only occur in a predefined time window of, say, 1-10min. If the transaction is not confirmed in this time window, it becomes invalid. This measure is to help prevent DOS and keep the backlog small. Normally DLT systems shop all transactions until they are confirmed, but this becomes complicated when a transaction is not confirmed within the time frame and can wait for days for confirmation. This action will result in a transaction either being confirmed immediately or cancelled. In this case, the wallet software has to resend the transaction. The system is designed for low-latency transactions, which makes sense.

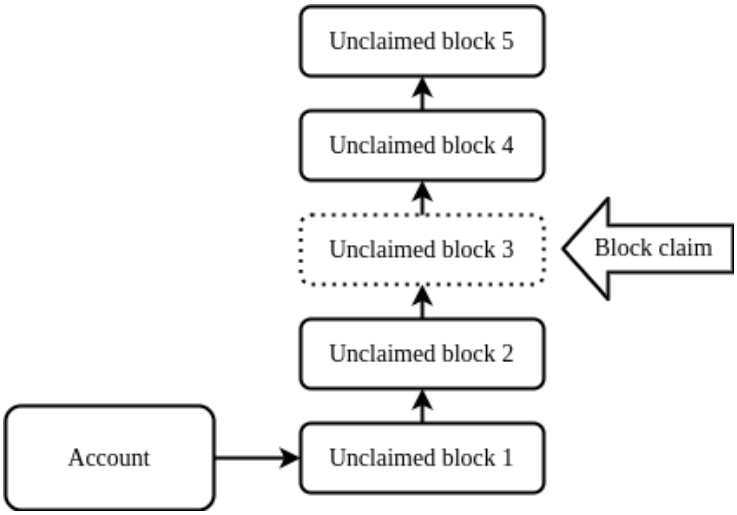

Figure 4: Unclaimed block queue.

I. Unclaimed block queue

Since not all nodes are online all the time and accounts can move from one node to another. It is difficult to require second level systems to monitor incoming/unused transactions sent to that account. For this, Flink implements an unclaimed block queue, shown in Figure 4, which allows an account owner to always correctly list all unclaimed blocks. The queue is updated in a separate transcation and is limited in size. Account holders must receive transactions on a regular basis, otherwise they risk not being informed of incoming transactions. Even if this information is missing, new transactions can be made at any time.

IV. MISCELLANEOUS FEATURES

Flink has some advanced features not normally found in other DLT projects. The functions are mostly modelled after existing banking systems, which makes the transition to a DLT system easier.

A. Payment Request

Flink introduced something usually found in newer traditional mobile banking solutions. Where a user can send a payment request to another user. That user then usually just confirms the payment with the amount, reference number and description already stored. In Flink, a payment request is sent to a node, which then redistributes it to all other nodes online. The wallet system for that user receives the request from the relying node and stores it in the local database. The nodes do not store payment requests, and the entire system operates only on the best-effort principle. This means that the wallet system and its node must be online for the request to be received.

B. Quantum Safety

With the development of quantum computers, quantum security is becoming increasingly important. Elliptic curve cryptography is vulnerable to quantum-based attacks, and quantum-resistant cyphers are not yet widely and fully tested. For this reason, Flink was designed with cyphers interchangeability in mind. For example, account addresses are not

created from public keys, but are randomly generated. The currently valid public keys are specified with the last confirmed block. This makes it easier to exchange keys and, in the future, also cyphers.

C. Reference Number/Code

The traditional banking system typically uses payment reference numbers to identify individual payments. With DLT, each payment is usually sent to a separate address. Because of Flink's account-based design, an additional payment reference code has been added to the block. This is a practical approach to tracking payments that is usually missing in DLT concepts.

V. ATTACK PREVENTION

The most problematic attack point of a feeless payment system are DOS (Denial of Service) and DDOS (Distributed Denial of Service)(Lau *et al.* [13]) attacks. Since transactions are free, the system is very vulnerable to spam transactions that overload the entire system. In Flink, many parts of the system are specifically designed to prevent such attacks.

A. DOS Attacks

The following describes the most common DOS attacks to which the system is most vulnerable. DOS Attacks are difficult to completely eliminate, especially in a system without fees. However, the system has many approaches to counter such attacks. Overall, according to the CAP theorem, the system prefers consistency and partition tolerance over availability. However, availability is potentially sacrificed only during the active attack phase.

1. Node Overflow Attack

In this attack, the entity creates multiple node addresses. Since all nodes must be confirmed before they are added to the database, this can place an extreme load on the entire system. The system protects itself by limiting the number of nodes that can be created per

day (adjustable parameter). With this protection, the system will function normally under these conditions.

2. Account Overflow Attack

This attack floods the network with multiple accounts. Each node can only publish a limited number of blocks (including account-creating blocks). For this attack to succeed, the attacker must have access to a node with a very high weight, since the publication rate depends on the node's voting weight. Even then, the rate is capped, so the probability of completely disabling the entire network is low. Also, high weight nodes are usually managed by crypto exchanges, which are likely to monitor the integrity and status of the nodes.

3. Transaction Overflow Attack

In this attack, the network is overrun with multiple transactions. As in the previous case, each node can only publish a limited number of blocks (including account creation blocks). For an attack to be successful, the attacker must have access to a node with a very high weight, which is not easy to achieve. Additionally, PoW might be introduced in the future to further prevent transaction spam.

4. Unconfirmed Transaction Overflow Attack

This attack targets each account's unconfirmed transaction queue. Accounts are required to confirm all incoming transactions. Once the queue reaches a certain size, new items are discarded. This makes it very difficult to actually compromise the system. The account holder can still receive transactions that aren't in the queue, but must manually retrieve them.

5. Brute Force Attack

In the event of a DDOS attack, nodes may be removed from the network. Such an attack could cause the system to become temporarily unavailable for confirmations, as the majority

voting weight wouldn't be reached. The system would recover quickly once the attack is over.

B. System Integrity Attacks

In these attacks, the attacker is usually trying to gain a financial advantage. Usually, they try to double-spend or something similar. The consensus algorithm ensures the integrity of the system. All transactions must be confirmed within a time interval with $\left[\frac{W_T}{2} + 1\right]$ voting weight. For the system to be compromised, someone would have to receive a majority voting weight. In practice this weight is set to a higher number to further increase security.

1. Primary Protection

Since $\left[\frac{W_T}{2} + 1\right]$ voting weight is quite large, the attacker would have to make large investments to gain that much voting weight, which is unlikely. Moreover, such investments would destroy the network and cause the investment to expire.

2. Fork protection

Local forks can occur when an account publishes two transactions with the same previous block. In this case, both transactions are voted on, but each node can only vote for one of the transactions. The transaction that reaches $\left[\frac{W_T}{2} + 1\right]$ will be persisted. If neither transaction reaches a majority within the specified time interval, both transactions are discarded.

VI. IMPLEMENTATION

In the implementation phase, the decision was made to build the system in a higher level (managed) programming language. Therefore, C, C++ were not considered. In the author's opinion, the system closely resembles a IT system with large memory and database requirements. Such systems are usually limited in performance by the database design and not by the programming language chosen. Therefore, the reference implementation was written in Java. The implementation is available on Github as an open source project under the Apache license.

A. Network

The system uses protobuf to serialize messages over a TCP stream. Connections between hosts are limited to one, which is a full duplex. Connection handling is asynchronous with low memory requirements, allowing multiple connections.

B. Storage

RocksDB key value store is used for storage. RocksDB is a highly scalable and performant database with transaction support developed by Facebook.

C. Cryptography

For signing, ED25519 (Bernstein *et al.* [14]) is used with standard SHA-512 hashing. Key derivation is based on BIP32 with ED25519 modification in SLIP-0100.

VII. CONCLUSION

Here we present a distributed feeless payment system with fast transaction confirmation time. The system is based on a voting-based consensus mechanism with a block lattice data structure. Since it does not involve mining, the system is highly energy efficient and can be operated on modest hardware. The system can be operated by the community with minimal investment while being secure and reliable.

References

[1] C. Scardovi, *Restructuring and Innovation in Banking* (Springer, 2016).

[2] R. Maull, P. Godsiff, C. Mulligan, A. Brown, and E. Kewell, Strategic Change **26**, 481 (2017).

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

[4] C. LeMahieu, "Nano: A feeless distributed cryptocurrency network," (2015).

[5] A. Devarajan and E. Karabulut, "Directed acyclic graph based blockchain systems," (2020).

[6] S. Popov, *IOTA: Feeless and Free*, Tech. Rep. (2019).

[7] A. Churyumov, "Byteball: A decentralized system for storage and transfer of value," (2016).

[8] K. Li, H. Li, H. Wang, H. An, P. Lu, P. Yi, and F. Zhu, Frontiers in Blockchain **3**, 11 (2020).

[9] E. Zhang, "A byzantine fault tolerance algorithm for blockchain." White paper. (2014).

[10] R. Ashikur, W. Olesinski, and P. Gburzynski (2004) pp. 73 – 78.

[11] Y. Guo, R. Pass, and E. Shi, *Advances in Cryptology - CRYPTO 2019*, Lecture Notes in Computer Science, **11692**, 499 (2019).

[12] B. W. Diack, S. Ndiaye, and Y. Slimani, International Journal of Advanced Science and Technology , 01 (2013).

[13] F. Lau, S. Rubin, M. Smith, and L. Trajkovic, in *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0*, Vol. 3 (2000) pp. 2275–2280 vol.3.

[14] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, in *CHES*, Lecture Notes in Computer Science, Vol. 6917 (Springer, 2011) pp. 124–142.