Asynchronous Blockchain without Consensus

Ramu Ramasamy

Abstract The strength of the blockchain network depends on its capacity to bear faulty nodes. Consensus mechanisms are expensive and it is not needed at all for the applications like cryptocurrencies. This report discusses a new blockchain architecture called Asynchronous Blockchain without Consensus (ABC) where the consensus mechanism is relaxed which is advantageous for applications like cryptocurrencies without proof-of-work. ABC model is completely asynchronous. This model does not support smart contracts as it requires consensus for functioning. ABC is fast, resilient to double-spending attacks and permanent. Furthermore, this report briefly describes two other blockchain models which emphasize on the fact that the wasteful proof-of-work can be eliminated and the consensus is not required for applications like cryptocurrencies.

Ramu Ramasamy
Paderborn University, e-mail: ramasamy@mail.uni-paderborn.de

1 Introduction

Bitcoin was the first decentralized digital currency and was backed up by Nakamoto Consensus protocol. A consensus protocol is expected to consider physical factors like network connectivity and size. The consensus protocol of blockchain targets the complete transaction history of the network [5]. The Nakamoto consensus protocol satisfies the conditions of finality, agreement, validity and integrity. The bitcoin architecture is permissionless as it allows participants to join and leave the system anonymously. Initially, the bitcoin's consensus was backed up by the Proof-of-Work (PoW) mechanism. In PoW, the security of the network completely relies on the hardware as it demands heavy computations. Therefore, PoW imposes heavy investments in setting up sophisticated hardware and running them. The participants of the network who contribute to maintain the blockchain system are called miners. The miners setup and maintain the hardware for PoW and receive compensations for adding the transactions of their choice to the block. PoW is a lottery based system and miners used higher hash rate to win the lottery [6]. The compensation is a percentage of the transaction value and usually miners prefer transactions with higher transaction value. The participants have to wait for the transactions to get completed. Transactions that are not mined for a long time get discarded. Major issues with PoW are its excessive energy wastage to find a valid solution for the computational problem. The entire network can be controlled by a miner if the miner acquires the majority (more than 50%) of the network's hash rate [6]. In addition, PoW suffers from communication issues between the miners regarding loss of message and timing guarantees [4].

Researches are going on replacing PoW with Proof-of-Stake (PoS) in Bitcoin with a target to save energy and make the network more secured. In PoS, the participants of the network are able to contribute to the system only based on the amount of cryptocurrency they hold. The participants deposit a stake in the competition and a participant with a higher stake wins the competition [5]. In contrast to PoW, a participant has to acquire more than 50% of the cryptocurrency tokens to gain the control of the network, which is not easy. The problem with PoS is it relies on randomness and participants has to collectively and repeatedly choose the leaders [4]. Similar to PoW, participants have to wait for the transactions to get completed.

This report describes a mechanism where consensus is relaxed and an asynchronous blockchain is introduced in order to increase the efficiency of the cryptocurrency blockchain along with other advantages.

2 Background

Blockchain is a list of blocks growing continuously. Blocks comprise of records of transactions. The blockchain is a decentralized, immutable digital ledger where each block will have a cryptographic hash of its previous block, a timestamp, and the data (amount in the case of cryptocurrency).

2.1 Permissionless Blockchains

A permissionless blockchain is a blockchain where the nodes can join or leave the network without any authorization. There is no central authority which can manage the memberships or controls the access rights or block the adversarial participants. Bitcoin and Ethereum are examples of permissionless blockchains. Permissionless blockchains are also called as public blockchains. Here, the block proposal usually depends on the computation power of the node, possession of cryptocurrencies, etc.

2.2 Permissioned Blockchains

Permissioned blockchains possess a mechanism for authorization of the nodes that can participate in the blockchain network. The nodes are required to be authorized in order to participate and contribute to the blockchain. There exists a central authority that decides on access rights and the privileges to the participants. Hyperledger Fabric and R3 Corda serve as examples for permissioned blockchains.

2.3 Network Synchrony

Network synchrony defines the degree of coordination required between the participants in the network.

- *Synchronous*: The operations performed by the participants are executed in rounds, and in each round, the same kind of operations take place. This can be achieved by incorporating a centralized clock service.
- *Asynchronous*: There is no centralized clock service, and therefore, the operations of the participants are not coordinated. There is no guarantee that the message transmitted is delivered and also there is no upper bound for the delay in communication between different participants.
- *Partially Synchronous*: Similar to an asynchronous network, the operations of the participants are not coordinated, but there exists an upper bound for the delay in communication between the components. The message delivery is guaranteed.

3 The Need for Consensus

A consensus mechanism is needed for a blockchain system in order to ensure all the agents in the blockchain agree on a single chain of transactions. A consensus protocol has a set of rules for message passing and processing of all the nodes available in the network to reach an agreement on the blockchain. The strength of the blockchain network can be determined by its capacity to bear the adversarial nodes. If a consensus protocol can tolerate a at least a crash failure then it is called Crash-fault Tolerant (CFT). If the consensus protocol is able to tolerate at least on Byzantine failure then it is called Byzantine Fault Tolerant (BFT).

3.1 Nakamoto Consensus

In Nakamoto Consensus [1], the block that extends the longest chain (chain with most proof-of-work) is accepted. When a new transaction is broadcasted to all the nodes, each node will add it to a block. Then, each node finds a proof-of-work for its block by solving a complex computational task. When the proof-of-work is found, the block is broadcasted to all the other nodes in the network. The other nodes accept the block only if the transactions in the block are not spent already. The acceptance is indicated by working on the next block for the received block. In order to propose a block, the miner should solve a complex computational task which consumes time and huge amount of electricity. Proof-of-Work is a mechanism that enhances the sybil resistance. When a single miner or a group of miner holds the 51% of the networks computing power, then the control of the blockchain network can be taken over by the miners. This makes double spending attack possible. The miner or group

of miners having the control, can erase a transaction in the blockchain and build a new chain by rebuilding the blocks.

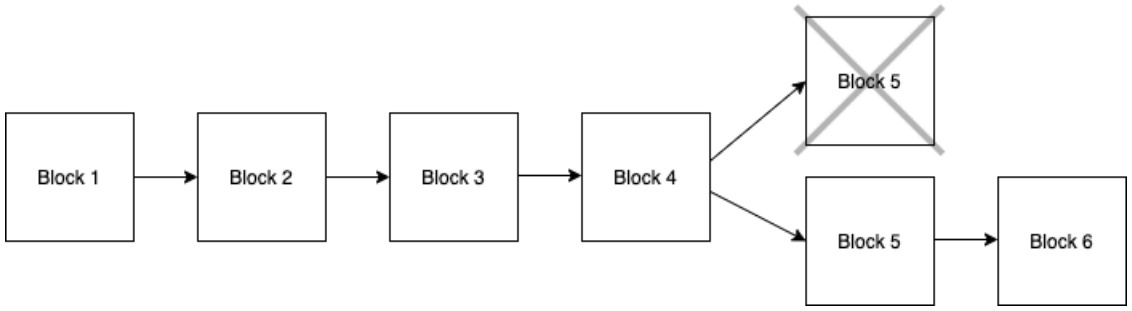

Fig. 1 Nakamoto Consensus and Proof-of-Work

4 The Idea of Alleviating Consensus

ABC is asynchronous, permissionless, final, deterministic, and simple. The transactions that are confirmed are irreversible. Unlike proof-of-work, the blockchain system does not depend on the computing power or other resources like energy spent on maintaining the blockchain. Instead, more than two-thirds of the amount of cryptocurrency should be held by honest participants.

A cryptocurrency blockchain should be robust to handle malicious transactions. Honest agents are the nodes that are behaving truthfully, and the agents that are misbehaving are called Byzantine nodes. A consensus protocol plays a vital role in handling double-spending. Person A has only one cryptocurrency coin and makes a transaction, sends one cryptocurrency coin to B. Simultaneously A is creating another transaction sending the same cryptocurrency coin to C which should be addressed.

A Byzantine Fault Tolerance (BFT) Consensus is defined by Agreement, Validity, and Termination. A BFT can tolerate a certain number of faulty nodes (Byzantine nodes). The mechanism has a set of validators equal to N. The validators are nodes in the blockchain system that validates the transactions created by other nodes based on the available information and vote their opinion regarding the same. For any blockchain network that has N nodes and f nodes being Byzantine, $N \geq 3f + 1$ is required to ensure consensus. The nodes can be partitioned into three equally sized groups with one group containing the faulty ones. Therefore, the blockchain network should have at least four nodes in order to tolerate one Byzantine node. ABC Consensus has the same definition for Agreement, and Validity.

Definition 1 (Agreement). [4, p. 2] "If some honest agent accepts a transaction, every honest agent will accept the same transaction. No conflicting transactions are accepted".

Definition 2 (All-Same-Validity). [4, p. 2]
"If the first transaction every honest agent sees is the same, this transaction is accepted by honest agents".

If A is truthful, A would have made only one of the two transactions, but A is not truthful here due to double-spending. In this scenario, some of the honest agents see one transaction first, and the other honest agents see another transaction first. Therefore, the definition of All-Same-Validity (Definition 2) fails but any of the transaction is accepted in a finite time (Definition 3), whereas,

Table 1 Termination in Consensus vs Termination in ABC Consensus

Termination (Consensus)	**All-Same-Termination (ABC Consensus)**
Definition 3. "Every honest agent accepts some transaction in a finite time" [4, p. 2].	**Definition 4.** "If the first transaction every honest agent sees is the same, this transaction is accepted by honest agents in a finite time" [4, p. 2].

in ABC, either of the transactions is not accepted, and the transactions stay forever without any actions. Ultimately, A ends up loosing the one cryptocurrency coin due to misbehaviour. Thus in ABC Consensus, honest agents accept non-conflicting transactions in a finite time, and conflicting transactions remain forever without any action.

The major operation that takes place in the blockchain is transactions. Every transaction is made up of a pair of keys. Each transaction is signed by a private key of the sender. Therefore, the transaction contains a signature, and the public key of the sender can be used to generate a signature that can be validated. Every transaction refers to at least one previous transaction and leads to the formation of a directed acyclic graph (DAG) [4, p. 2]. One agent delegates itself for making a transaction. Another set of agents called validators validates this transaction. In addition, in order to identify a validator, every transaction holds another public key that corresponds to the validator. The validators issue transactions called **acks**. The acks are nothing but 'dummy' transactions acknowledging and confirming the actual transaction [4, p. 3].

Let us consider a transaction t which is created by an owner with a set of inputs. This transaction t will only be confirmed when there are enough acks referring to it. In a case, where the owner misbehaves and creates another transaction t' almost at the same time with the same set of inputs (double-spending) then, there are two possibilities: (1) Either of the transactions t or t' is accepted and confirmed; (2) Both the transactions are not accepted, and this happens when some of the validators see acks referring to t, and some of the validators see the transaction t'. The second case will lead to a situation where transactions remain unaccepted forever, and the misbehaving agent loses the value of the transaction as a penalty for double-spending. This will not affect the blockchain in anyway and does not influence other transactions.

5 The ABC Model and Assumptions

The ABC model comprises of participants called agents, and there are two types of agents: (1) honest agents are agents who obey all the protocols of the blockchain; (2) adversaries are agents who do not follow the protocols of the blockchain. At any point in time, the adversaries cannot hold one-third of the network's cryptocurrency. The agents in the ABC model communicate with each other by broadcasting messages forming a virtual network. Like any typical permissionless blockchain, the agents can join the network anytime, broadcast messages and also can leave the network. The blockchain is assumed to be asynchronous where the messages are not time-bounded. It is only required that the messages sent are delivered to the recipient eventually [4]. Even though the adversary controls the network, it can only delay the time that takes for the message to reach the recipient, hindering the progress of the agent but it cannot take control or interfere with the protocols of the honest agents. The model is assumed to have asymmetric encryption. The message is encrypted by the receiver's public key and signed by the sender's private key. The receiver has first to decrypt the message using the sender's public key and then decrypts the message again using the receiver's private key. The model is also assumed to have cryptographic hashing and computes a unique hash.

6 Protocol and Components

This section describes the various components involved in the protocol [4, p. 3].

Outputs: Outputs are the fundamental unit of information and every transaction contains an output in order to identify the cryptocurrency holders and the validators. The output comprises:

- *Value*: The amount of cryptocurrency.
- *Owner key*: It is a public key representing the owner of the cryptocurrency holding the corresponding private key. The owner controls the output.
- *Validator key*: It is a public key representing the validator of the transaction holding the corresponding private key.

Genesis: Genesis shows the initial distribution of cryptocurrency among all the agents in the form of set of key-value pairs. The value represented by the genesis corresponds to amount of cryptocurrency or the stake held by the agent. Each key-value pair of genesis is called as output. The genesis transaction will have only outputs. The output can be controlled by an agent if that agent knows the private key corresponding to the key-value pair. We assume that the initial distribution of cryptocurrency is $3f+1$, every output can be uniquely identified with the associated private key, and one output is controlled only by one agent.

6.1 Transactions

Transactions are requests or messages broadcasted in the network. A transaction can be a spending transaction or a reference transaction ack. If an agent wants to send money to another agent, a message indicating the amount to be transferred will be broadcasted in the blockchain network. This value of the cryptocurrency that should go down in the broadcasting agent should be equal to the value of the cryptocurrency getting added to the receiving agent. The outputs of a transaction aids in identifying the owner and validator of the transactions. This validator can issues acks for the transaction. When a message is broadcasted through the blockchain network, a small number of validators contribute in sending acks to validate the transaction. If an agent wants to change the validator, the agent can trigger a transaction with a different validator. Furthermore, the agent can act as an owner as well as a validator for the same transaction. The blockchain network only keeps track of all the transactions occurred, and therefore, the transactions are linked to one another, and every transaction refers to a previous transaction. The set of transactions that can be referred from a transaction t is called as $past(t)$. Thus, all the transactions form a directed acyclic graph (DAG). This blockchain network works asynchronous and each agent might be aware of some of the transactions in the DAG.

Definition 5 (Spending Transaction). [4, p. 4]
A spending transaction t contains:

- A set of references to previous transactions.
- A set of outputs
- A set of inputs, where each input is an output of the transaction in $past(t)$. Transaction t is said to spend these inputs.

"The sum of values of the outputs is equal to the sum of values of the inputs, and called the value of t".

"The transaction is signed by inputs. The agent creating the transaction controls the inputs of the transaction".

Definition 6 (acks). [4, p. 4]

"An ack contains a set of references to the previous transactions. For some output o, a and signed by the validator key of o. The weight of a is the value of o and we say that a is associated with o".

As mentioned every transaction refers to the previous transaction. Every transaction has a unique hash by which the transaction is identified. This hash is used by the successive transaction for reference. The hash references to the previous transactions can never be cyclic and therefore, all the transactions form a DAG.

6.2 DAG Interpretation

When a transaction t is created, the validator has to check for the previous transactions $past(t)$. By doing so the validator can make sure if the transaction has valid references to the data. Therefore, a transaction t will only be processed and added to the block when the agent has the full record of transactions $past(t)$. This is because when a new transaction t is issued and when an agent receives t, it checks for $past(t)$. If some of the transactions in the $past(t)$ are missing, the agent cannot be sure that the transaction t is not referencing invalid transactions. Therefore, the agent requires a complete history of transactions in the past in order to process t. Figure 2 shows a DAG for spending transactions where p_i's are the owner keys and s_i's are the corresponding secret keys. The genesis shows the initial distribution of the amount among the nodes. The outputs of the genesis serve as the inputs for the spending transactions. $(p_1, 4)$ is an output of the genesis transaction that serve as the input for the spending transaction producing $(p_4, 2)$ and $(p_5, 2)$ as outputs. The transaction signed with secret keys s_3 and s_6 indicates that multiple spending transactions can occur together.

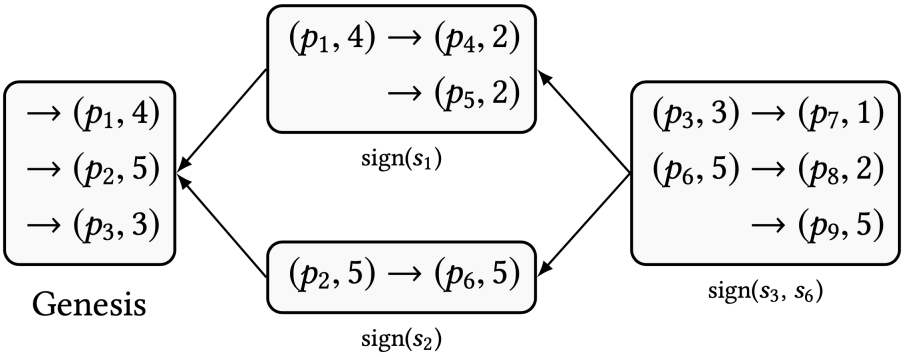

Fig. 2 Example DAG of Spending Transactions [4].

Definition 7 (Past). [4, p. 4] "The set of transactions reachable by following references t is $past(t)$. For a set of transactions T, $past(T) = \cup_{t \in T}\ past(t)$".

For a transaction t to happen, all the previous transactions $past(t)$ are required.

Definition 8 (Depends). [4, p. 4] "If a transaction v spends one or more outputs of transaction u, then v depends on u. Dependence is transitive and reflexive, i.e. every transaction depends on itself and if w depends on v and v depends on u, then w depends on u."

Definition 9 (Conflicts). [4, p. 4] "If two transactions u and v spend the same output, they conflict. Moreover, for every conflicting transactions u, v, every transaction that depends on u will conflict with every transaction depending on v."

Two transactions t and t' spending the same output that resulting in an inconsistent state of the blockchain network.

Transactions are confirmed only based on the set of transactions that refer them. When a transaction t is created, and when this transaction t is visible to the validators, the validators broadcast acks referring t. When there are enough validators $(2f + 1)$ referring t indicating that t was the first transaction observed by the validators, it gets confirmed. The confirmed t is irreversible. For a transaction to get confirmed, a set of acks C_t references t. If the total number of acks C_t referring t is more than the two-thirds of the amount or stake in the blockchain system, it indicates that the creator of the transaction t would not have misbehaved in $past(C_t)$, and then t gets confirmed. The acks C_t serves as a proof indicating that t is now confirmed.

Definition 10 (Effective Ack). [4, p. 5]
If an ack contributes to the confirmation of the transaction, we call it effective. An ack a associated with output o is effective for t if:

- The transaction outputting o is confirmed in $past(a)$.
- No transaction in $past(a) \setminus \{t\}$ spends o.
- Every unconfirmed transaction in $past(a)$ that t depends on is the only transaction in $past(a)$ spending its inputs.

Definition 11 (Confirmed). [4, p. 5]
Some transactions becomes confirmed depending on the DAG. Genesis is by default confirmed.

A transaction t is confirmed if there exists a set C_t of acks effective for t associated with different outputs that are unspent in $past(C_t) \setminus \{t\}$, such that the sum of weights of acks in C_t is at least $2f + 1$.

6.3 Creating Transactions

When a transaction t is created with an output by a honest agent, the same honest agent will not create another transaction t' with the same output conflicting with the transaction t. The honest agent creates an ack a_1 referring the transaction t and the same honest agent creates an ack a_2 that refers a_1 i.e., $a_1 \in past(a_2)$, and $t \in past(a_1)$. Also, when an honest agent creates a transaction t_1 first and t_2 next, then $t_1 \in past(t_2)$.

Algorithm 1 Creating Transactions

1 Transaction t references all previously observed transactions.
2 Broadcast transaction t to the network.

6.4 The Adversary

The behavior of the adversary cannot be predicted and hence arbitrary. The adversary might create a transaction t' having the same output of transaction t, thus conflicting with the other. Whenever a message has been broadcasted through the system, the adversary can gain access to the message immediately. As already discussed, the adversary cannot gain access in controlling the entire system but has the ability to delay the time taken for delivery of a message from one honest agent to another honest agent. The total value of all initial inputs in the blockchain sums up to $3f + 1$ where f is the maximum value that can be held by adversaries and $2f + 1$ of the amount is held by the

honest agents. This means that at any point, the adversaries can hold a maximum of one-third of the cryptocurrency available in the blockchain.

When a transaction is made by an adversary sending money to an honest agent, and the transaction has an output with a validator which is also an honest agent and, there exist acks referring this transaction then the value of the transaction is reduced from the adversary's holdings and the same value of the amount is increased for the receiving honest agent.

When a transaction is created by an honest agent sending money to an adversary, and the transaction has an output with an honest agent or an adversary as a validator then the value of the transaction is reduced from the amount held by the honest agent and the same value of the amount is increased for the adversary.

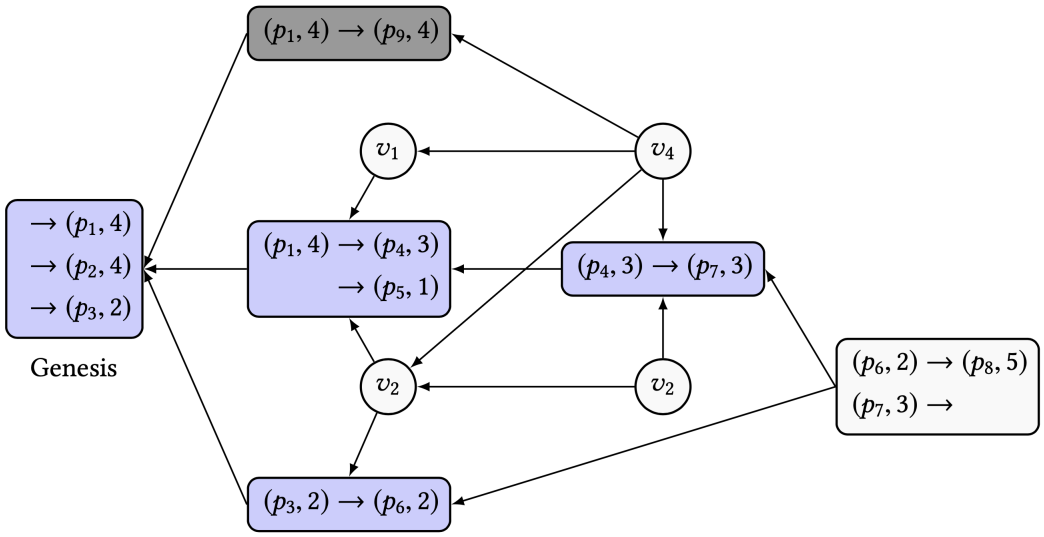

Fig. 3 Example Transaction DAG with Double Spending. p_i represents the validators of the output of each transaction. v_i represents acks created by the validators referring to the transactions. The spending transaction p_3 is confirmed by the acks v_2 and v_4. All the transactions represented with blue background are confirmed. The transaction with grey background (say t') is a conflicting transaction, trying to double-spend and this transaction will never be confirmed and remains forever in the system. The validator v_4 that refers the transaction t' finds another transaction t in the past that had already used the same output of t' and therefore, the ack created by v_4 referring t is never effective [4, p. 6].

7 Double Spending

Considering ABC Consensus discussed in Section 4, it is assumed that when a transaction t' is conflicting with a transaction t and, neither of the transactions are confirmed as per Agreement (Definition 1), then at least two-thirds of the cryptocurrency in the blockchain are held by the honest agents as per the discussion in section 6.4. Therefore, when there is no conflicting transaction for t, then the All-Same-Validity (Definition 2) and All-Same-Termination (Definition 4) hold.

Corollary 1. *If Agreement holds, ABC protocol satisfies ABC consensus [4, p. 5].*

In order to create a contradiction, two transactions t and t' conflicting with each other by spending the same outputs are created and confirmed. Let G be an instance of DAG, that is minimal in terms of the number of spending transactions.

Lemma 1. *If a confirmed transaction u depends on v, v is confirmed [4, p. 5].*

Proof. u is a confirmed transaction. Therefore, as per Definition 10, transaction u will have a set of effective acks C_u. Let a is an effective ack of u, then $u \in past(a)$ holds. Since, the transaction u depends on the transaction v, then $v \in past(u)$. As per transitive dependency, any ack $a \in C_u$ also denotes, $v \in past(a)$. Futhermore, this indicates that preconditions of effective acks (Definition 10) for v are subset of preconditions of effective acks for u. Thus, by Definition 11 v is confirmed.

Lemma 2. *There are no unconfirmed transactions in G [4, p. 5].*

Proof. Let us consider there exists an unconfirmed transaction u in G. As discussed in Lemma 1, if a transaction u is not confirmed, no other transactions can depend on u. u can have an ack a referring to it only when it is confirmed (by Definition 10). Another instance of DAG G' is obtained by removing the transaction u and all the transactions that depend on u. Then, no effective acks can be removed from G as they refer confirmed transactions. Now, G' has all the transactions in G that are already confirmed, including t and t'. Therefore, "any transaction confirmed in G is also confirmed in G'" [4, p. 6].

Lemma 3. *Suppose t is the first transaction confirmed in G during the execution of the protocol. Then, no transaction conflicting with t can become confirmed [4, p. 6].*

Proof. Let us consider t is the first transaction confirmed. Therefore, by Definition 10, the transaction t will have a set of effective acks C_t. As discussed in the Section 6.4 at any point, the value of initial outputs for which the adversary is the validator can hold up to f. Thus, there should be at least majority of honest agents $(f + 1)$ should have sent acks referring the transaction t. Let V be the number of honest agents that act as validators, sending acks a that contribute to effective acks C_t to t. Then, $t \in past(a)$. Thus, these V validators cannot send acks for another transaction t' that conflicts with t by spending the same inputs as t. By Definition 11, there are no transactions $u \in past(C_t) \setminus \{t\}$ spending outputs associated with V [4]. Thus, any transaction can receive effective acks a when $t \notin past(a)$ of cumulative weight of $2f$.

Corollary 2. *There is no transaction conflicting with t in G.*

Lemma 4. *Let t be the first transaction confirmed in G. There is a DAG G' where the set of confirmed transactions is the same except not including t, where the genesis contains the outputs of t but does not contains the inputs of t [4, p. 6].*

Proof. Consider two validators V_o and V_i which sends effective acks for outputs of t and inputs of t respectively. Consider a confirmed transaction u and recall from Lemma 3 that either $u \in past(C_t)$ or $t \in past(C_u)$. The validator V_i issued acks for u that contributed to effective acks C_u. t is a spending transaction and is spending the output of V_i and therefore, V_i would have issued the effective acks to u only when $t \notin past(C_u)$. Therefore, $u \in past(C_t)$ is true. When the validator V_o sends an effective ack a, then $u \in past(a)$. This indicates that when V_i and V_o sends effective ack for the transactions t and t' respectively, these transactions cannot conflict. Hence, the transaction t can be added to the genesis of G' and the validators can issue acks referring t. The value of input and outputs of the transaction t may not be same in their values. In such case, the smaller inputs and outputs in the genesis can be accumulated independently so that it can match tha value of t.

Theorem 1. *No DAG can be produced by the ABC protocol such that a pair of confirmed transactions are conflicting [4, p. 7].*

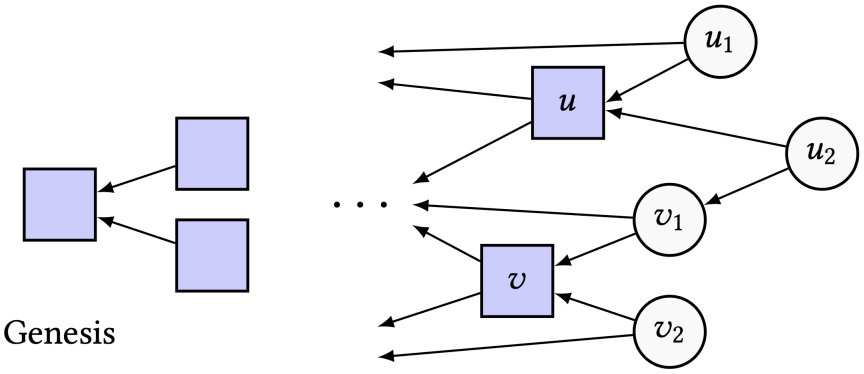

Fig. 4 Example illustration of $v \in past(C_u)$. C_u comprises of effective acks u_i and C_v comprises of effective acks of v_i. If both the transactions u and v are confirmed, then either $v \in past(C_u)$ or $u \in past(C_v)$ [4, p. 7].

Proof. Consider, the first transaction confirmed by protocol execution is t. As discussed no conflicting transactions are confirmed in G (Corollary 2). The DAG G has all the transactions that are confirmed. A DAG G' can be created with all the transactions as in G but without the inputs of t. Then by Lemma 4, t can be added to the genesis in G'. This is a contradiction to the minimality of G.

Corollary 3. *ABC protocol satisfies ABC Consensus [4, p. 7].*

8 Improvements

8.1 Transaction Fees

The subject to spamming attacks. Therefore, transaction fees can be applied for every transaction in order to prevent the system from spamming attacks, and motivate validators to maintain the system by providing an incentive. A simple fee structure for every transaction is provided by the author [4].

Since, acks are dummy transactions that does not contribute to any share in the amount held by the system but are used to refer the actual transactions, acks are has no fees. When a transaction is broadcasted in the system, a certain amount of acks has to be issued referring the transaction to prove the truthfulness of the transaction. Thus, for any transaction the system requires acks to be effective in order the transaction to be valid. To prevent ack spamming, the volume of acks broadcasted is decreased. Instead of selecting validators that can receive incentives for issuing acks, all the validators are eligible for a part of the transaction fee (incentive) based on the weight of the ack. For example, the issuer of ack a of weight w will be provided with an incentive of amount ϵ:

$$\epsilon = \frac{w}{3f+1} \sum_{u \in effective(a)} fee(u) \tag{1}$$

where $effective(a)$ is the set of transactions to which a is effective and $fee(u)$ is the transaction fee paid by u.

8.2 Money Creation

When the transactions fees are charged as per the Equation 1, the amount spent as transaction fee by the issuer of the transaction will be equivalent to the sum of amount collected as incentive for the transaction. Thus, the overall amount in the system remains the same over time. However, the system can be set up in such a way the amount of cryptocurrency increases over time by multiplying the fee ϵ with a constant. For example, the ϵ can be multiplied with a constant $\alpha > 1$.

9 Related Work

9.1 A Non-Consensus Financial Transaction Processing Model

This subsection describes a financial transaction model explains a permissioned blockchain that does not depend on a consensus mechanism [2]. Furthermore, the proposed design also relies on a designated set of validators to validate and confirm the transactions.

In Bitcoin, a transaction is a representation of mapping between inputs and outputs. Only transaction outputs that are unspent can be utilized as inputs for new transactions. Thus, the mapping of inputs and outputs acts as proof that the transaction creator has sufficient amount for creating a new transaction without even verifying the balance amount associated with the creator. As already discussed, in order to prevent the problem of double-spending, the agent accepts the chain that has most proof-of-work done, the longest branch of the blockchain.

This transaction processing model works similar to Bitcoin but with a difference where each server node will maintain a list of transactions that are processed by it and not all the transactions. Similar to *acks* in ABC, a set of server nodes validates a newly created transaction. Each server node sends validations to the newly created transaction irrespective of the validations sent by other server nodes in the blockchain network. In addition, the server node adds a signature for the transaction if it considers the newly created transaction as valid based on the information it holds. Similar to the Definition 11, a transaction is said to be confirmed when it receives valid signatures from $2f + 1$ server nodes. When a server node validates a transaction by signing it, then it has to maintain the transaction as a part of the transactions it holds and this transaction can be used in the future to validate other transactions. The confirmed transactions become permanent and represents the record of activity of the servers involved in the transaction as sender or receiver. This transaction processing model prevents the problem of double spending without a need for consensus protocols.

Table 2 Notations used in Section 9.1

Notation	Description
t	Proposed transaction
s	Server node initiating the transaction
id	Sequence number of a transaction
O	Set of redeemable outputs
I	Set of authorized input transactions
d	Digital signature of the sender node
t_p	The most recent transaction before t
s_p	Server node processing t
T^A	Local set of authorized transactions in s_p

9.1.1 Life Cycle of a Transaction

Every transaction has two life cycle stages:

1. *Proposed Transaction*: A transaction t that is initiated by a server node representing a sender. Every proposed transaction has five components: s, id , O, I, and, d.
2. *Authorized Transaction*: If a server node considers the transaction t as valid, it adds its signature to the transaction. t is considered as an authorized transaction if it receives at least $2f + 1$ signatures from the blockchain network.

9.1.2 Transaction Validations

Every transaction has a sequence number for ensuring that the processing of the transactions happens in a sequence manner. When a transaction (t) is proposed by a server node, this transaction is broadcasted to all the other server nodes for authorization. Along with the proposed transaction, the most recent transaction (t_p) initiated by the same sender (which is authorized previously) is also sent. Every server node has a local set of authorized transactions. In addition, a proposed transaction t has a set of authorized input I transactions that can be used by other server nodes for validating t. If s_p processing t does not has the authorized transactions in T^A that are required to validate t then s_p can make use of the set of authorized transactions (I) available with t.

A transaction t is considered valid by honest server nodes only when the transaction components are structurally and functionally valid. The following conditions should hold.

- **Structural Validity**

  1. t will be accompanied by t_p. The sequence number of t_p should be one less than the sequence number (id) used in t.
  2. All the input transactions in I of t should be authorized transactions.
  3. The sum of amounts in I should be equal to the sum of amount in outputs O.
  4. The signature d should be a valid for the sender for the server node s.

- **Functional Validity**

  1. The sequence number of t should be greater than the sequence numbers of all the authorized transactions available in the local authorized transaction set (T^A) of the processing server node (s_p).
  2. All transactions in I should be unspent according to T^A available in s_p that is processing t.

If a transaction is structurally valid according to a honest node, it will be structurally valid according to all the other honest nodes. However, functional validity of a transaction may vary among the honest nodes as it depends on the local authorized transaction set (T^A). A transaction is authorized only if it is structurally and functionally valid. A transaction becomes functionally invalid if the proposed transaction is a double-spending transaction.

Let us consider there exists a transaction t initiated by a server node s. t has used an input. Now, a new transaction t' is proposed that uses the same inputs as of t (double-spending). If the sequence number of t is greater than or equal to the sequence number of t', then the t' becomes functionally invalid and is not authorized. Alternatively, if the sequence number used in t is lesser than the sequence number used in t', it means that t is an authorized transaction that was validated by $2f + 1$ server nodes prior to the functional validation of t'. This means that t' will be detected as an attempt of double-spending by at least $f + 1$ honest server nodes and can be signed by at most $2f$ nodes which is not sufficient for authorizing t'. Hence, if a transaction is authorized, it cannot be a double-spending transaction.

9.2 The Consensus Number of a Cryptocurrency

This section describes that the consensus number of a cryptocurrency like Bitcoin is 1 [8] which shows that consensus is not required in order to prevent double-spending. There are two models discussed: (1) shared memory model, and (2) message passing model. The consensus number is studied in shared memory model. An asset transfer system comprises of a set of accounts and each account has an associated owner process. Amount from the accounts can be withdrawn only by the associated owner processes. However, the balance amount of an account is transparent to every process. The main insight here is that linking accounts to specific owner processes eliminates the need for consensus. Only the owner process decides on the order in which the amount has to be transferred without any involvement of other processes. However, other processes validate the transfer made by the associated owner process which is similar to *acks* in ABC. Every owner process is associated with a distinct location in the atomic snapshot memory. All the incoming transfers are stored in the atomic snapshot memory. In order to validate the withdrawal amount of a transfer these incoming transfers stored in the atomic snapshot memory are utilized. Intuitively, only one withdrawal can be active on an account at any given time. If the withdrawal transaction is successful, then it is added to the atomic snapshot memory. Furthermore, multiple processes are allow to decide on a withdrawal of amount from the same account. This is called a $k-shared$ asset transfer object where k number of processes are deciding on a transfer which also results in the consensus number being k.

As already discussed, every process points a specific location in the atomic snapshot memory where it stores all the successful transfer operations executed by the it. When each account is owned by a single process, all the incoming transfers of the account are stored in the location associated with the owner process. In order to determine the balance of an account, the process takes the initial balance amount along with the incoming transfers and outgoing transfers performed on the account. In order to perform a transfer, the owner process first determines the balance amount. If the amount to be transferred is lesser than or equal to the balance amount, then the transfer is processed by the owner process and the transfer is recorded in the list of transactions in the snapshot location associated with the owner process. Let us consider two processes (p and q) acts as shared owners of an account. The process p and q creates a transfer from account a to another account. The amount to be withdrawn from the account a is chosen specifically such that only one transfer can ever succeed, and, if the transfer is succeed, the remaining balance will uniquely identify either of the processes p or q depending on whose transfer was succeeded. If the account has only a sufficient balance for one of the transfers, then only the first transfer operation applied to the object is succeeded and the transfer initiated by the other process has to fail due to insufficient balance.

Similar to the transaction processing model described in Section 9.1, the message passing model implements a property called *account order* which is nothing but embedding sequence numbers to the transfers. If a honest process p delivers a message (transfer) with m with a sequence number s, and message m' with a sequence number s' such than m and m' are associated with the same account and $s < s'$, then m is delivered before m' by the honest process p. Similar to ABC, and the model discussed in Section 9.1, when a sender sends a message, the message is broadcasted to all the processes and waits until all it receives acknowledgements from at least $2f + 1$ processes. A message m with sequence number s will be acknowledged by a honest process only when the sequence number of the last message s' associated with the account a should be one less than the sequence number of the current message m ($s' = s - 1$). When the owner of the account sends conflicting messages for the same sequence number, then the account may get blocked. This indicates that even a compromised account is prevented from double-spending and no honest accounts get affected by malicious activity by compromised accounts.

10 Conclusion

This report described that consensus is actually not required in order to prevent double-spending. ABC is asynchronous, permissionless, final, simple, and permanent. ABC is suitable and advantageous for applications like cryptocurrencies but applications like smart contracts require consensus and it is not possible to build a smart contract mechanism with it. This model provides the features with the full functionality of a cryptocurrency without a need for wasteful proof-of-work which requires heavy investments. Section 9 described two different models but working in a very similar approach as ABC emphasizing on the fact that consensus is not required in order to prevent double-spending and finality.

References

1. Satoshi Nakamoto: Bitcoin: A Peer-to-Peer Electronic Cash System, pp.2–3, www.bitcoin.org (2008)
2. Saurabh Gupta: A Non-Consensus Based Decentralized Financial Transaction Processing Model with Support for Efficient Auditing, pp.16–38 (2016)
3. Karl Wüst, Arthur Gervais: Do you need a Blockchain?, Crypto Valley Conference on Blockchain Technology, pp.1–2 (CVCBT) (2018)
4. Jakub Sliwinski, Roger Wattenhofer: ABC: Asynchronous Blockchain without Consensus, arXiv:1909.10926 (2019)
5. Sachin S. Shetty Charles A. Kamhoua Laurent L. Njilla: Blockchain for Distributed Systems Security, pp.25–42, IEEE Press (2019)
6. Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHun Nyang, and Aziz Mohaisen: Exploring the Attack Surface of Blockchain: A Systematic Overview, pp.8–9, arXiv:1904.03487v1 (2019)
7. Yang Xiao, Ning Zhang, Jin Li, Wenjing Lou, Y. Thomas Hou: Distributed Consensus Protocols and Algorithms, pp.2–5, Wiley-IEEE Press (2019)
8. Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovi, Dragos-Adrian Seredinschi: The Consensus Number of a Cryptocurrency, arXiv:1906.05574v1 (2019)