ABC: Asynchronous Blockchain without Consensus

Jakub Sliwinski and Roger Wattenhofer
{jsliwinski,wattenhofer}@ethz.ch
ETH Zurich

ABSTRACT

We relax the definition of consensus for the purpose of an efficient cryptocurrency and introduce a new blockchain architecture called ABC that offers the typical functionality of a cryptocurrency. By foregoing the assumption of establishing consensus, we are able to design ABC with an array of advantages compared to typical blockchain protocols: ABC is permissionless, deterministic, and resilient to complete asynchrony. ABC features finality and does not rely on wasteful proof-of-work.

Without establishing consensus, ABC cannot support certain applications, in particular smart contracts that are open for interaction with unknown agents. However, our system is an advantageous solution for many important use cases, such as cryptocurrencies like Bitcoin.

1 INTRODUCTION

Nakamoto's Bitcoin protocol [10] has taught the world how to achieve trust without a designated trusted party. The Bitcoin architecture provides an interesting deviation from classic distributed systems approaches, for instance by using proof-of-work to allow anonymous participants to join and leave the system at any point, without permission.

However, Bitcoin's proof-of-work solution comes at serious costs and compromises. The security of the system is directly related to the amount of investments in designated proof-of-work hardware, and to spending energy to run said hardware constantly, forever. Since the system's participants called miners bear significant costs, the protocol compensates them with Bitcoin for running this hardware and spending energy. However, adversaries might disrupt this scheme by bribing the miners to behave untruthfully or disrupt the reward payments.

To make matters worse, proof-of-work protocols assume critical requirements related to the communication between the participants regarding message loss and timing guarantees. In other words, such protocols are vulnerable to attacks on the underlying network.

In the decade since the original Bitcoin publication, researchers have tried to address the wastefulness of proof-of-work. One of the most prominent research directions is replacing Bitcoin's proof-of-work with a proof-of-stake approach. In proof-of-stake designs, miners are replaced with participants who contribute to running the system according to the amounts of cryptocurrency they hold. Alas, proof-of-stake protocols require similar communication guarantees as proof-of-work, and thus can be attacked by disrupting the network. Moreover, proof-of-stake introduces some of its own problems. Prominently, existing proof-of-stake designs critically rely on randomness. To achieve consensus, the participants of such systems repeatedly choose a leader among themselves. Despite being random, this choice needs to be taken collectively and in a verifiable way, which complicates the problem.

Due to the way blockchains typically process transactions, participants have to wait significant amount of time before they can be confident that their transactions are accepted by the system. For example, it usually takes upwards of an hour for merchants to accept Bitcoin transactions as confirmed, which is unacceptable for time-sensitive applications.

In his seminal paper, Nakamoto made the crucial assumption that his system has to be able to totally order the transactions submitted to the system in order to reject the fraudulent ones. However, meeting this requirement is equivalent to solving the problem known in computer science as consensus. Nakamoto's assumption has shaped the design of blockchain systems to this day. Thus, many blockchain systems achieve consensus while not taking advantage of this powerful property, but suffering the associated costs.

Our Contribution. We relax the usual notion of consensus to extract the requirements necessary for an efficient cryptocurrency. Thus we introduce an asynchronous blockchain design that features an array of advantages compared to alternatives. In other words, we present an Asynchronous Blockchain without Consensus (ABC).

Asynchronous: ABC does not require the messages to be delivered within any known period of time. Thus ABC is fully resilient to all network-related threats, such as delaying messages or network eclipse attacks. An adversary having complete control of the network can halt progress of the system (by simply disabling communication), but cannot interfere with the protocol or trick the participants in any way. Previously approved transactions cannot be invalidated and impermissible transactions cannot be approved.

Permissionless: ABC is permissionless in the same way as existing proof-of-stake systems. Participants of ABC

holding exchangeable cryptocurrency take part in approving new transactions.

Final: If the network communication is not disturbed, in ABC transactions are instantly confirmed. Confirmation is final and impossible to revert, thus making ABC a suitable solution for time-sensitive applications. This is in contrast to systems such as Bitcoin, where the confidence in a transaction being accepted only increases with the passage of time.

Deterministic: We assume the functionality provided by asymmetric encryption and hashing. Apart from these cryptographic necessities, ABC is completely deterministic and surprisingly simple.

Proof-of-stake: Unlike proof-of-work, the security of the system does not depend on the amount of devoted resources such as energy, computational power, memory, etc. Instead, similarly to other proof-of-stake protocols, ABC requires that more than two thirds of the system's cryptocurrency is held by truthful participants.

Many important applications, such as cryptocurrencies, do not require consensus [4], and ABC offers an advantageous solution for those scenarios.

On the negative side, not being able to establish consensus in the traditional sense prevents some more general applications from being feasible. Such applications involve smart contracts open for interaction with previously unknown agents. For example, the functionality of Ethereum cannot be fully implemented with ABC.

2 RELAXING CONSENSUS

A cryptocurrency needs to be resilient to some of the agents behaving maliciously. The problem of establishing consensus in such an environment is called Byzantine agreement. The agents behaving truthfully are called honest, and malicious agents are called Byzantine.

In the context of a cryptocurrency like Bitcoin, consensus is used to solve the problem of double-spending. Suppose Alice holds one cryptocurrency coin. Then Alice broadcasts a transaction transferring her coin to Bob. Simultaneously, Alice tries to cheat and broadcasts a transaction transferring her coin to Carol. Upon receiving one of Alice's transactions, honest agents need to decide what happens to Alice's coin in agreement, preventing Bob and Carol from being tricked into thinking otherwise. In this context, according to the usual definition, achieving consensus consists of the following requirements:

Definition (Consensus).

Agreement: *If some honest agent accepts a transaction, every honest agent will accept the same transaction. No conflicting transactions are accepted.*

All-Same-Validity: *If the first transaction every honest agent sees is the same, this transaction is accepted by honest agents.*

Termination *Every honest agent accepts some transaction in a finite time.*

The key insight leading to the relaxation of this definition, is that if Alice misbehaves, it is a valid course of action to ignore her transactions altogether. If Alice behaved truthfully, she would issue only one of the conflicting transactions. Thus, every honest agent would see the same transaction first. Hence we relax the consensus requirement in the following way:

Definition (ABC Consensus).

Agreement: *As above.*

All-Same-Validity: *As above.*

All-Same-Termination *If the first transaction every honest agent sees is the same, this transaction is accepted by honest agents in a finite time.*

Under this relaxed notion of consensus, a cheating Alice might end up not sending her coin the either Bob or Carol. Some honest agents might see one of the transactions first, while others might see the other first. Then our requirement of All-Same-Termination does not apply, and the transactions might stay without a resolution forever. This turn of events can be seen as Alice losing her coin due to misbehaviour.

Otherwise Consensus and ABC Consensus do not differ. Agreed upon results are final, conflicting results are precluded and honest agents have their transactions accepted in finite time.

Surprisingly, despite the difference being so insignificant with respect to the functioning of a cryptocurrency, this relaxation allows ABC to combine an unprecedented set of advantages.

3 INTUITION

For simplicity of presentation, we describe ABC in the terminology of a cryptocurrency. A more formal description follows in Section 5. As usual in cryptocurrencies, the main operation is a transaction, which transfers cryptomoney from one or more inputs to one or more outputs. Inputs and outputs are cryptocurrency amounts paired with keys required to spend them. Every transaction refers to at least one previous transaction, such that all transactions form a directed acyclic graph (DAG).

An agent holding the cryptomoney of an output delegates said cryptomoney by indicating another agent devoted to maintaining the system called validator. To indicate the validator, outputs include a second public key. Validators can

issue 'dummy' transactions called acks referring other transactions to acknowledge and confirm them. Every agent can be indicated as the validator.

A transaction t is confirmed by the system if enough acks (directly or indirectly) refer to t, without referring to other transactions spending the same inputs. If a transaction receives enough support, no other transaction conflicting with t can become confirmed. In particular, if the owner afterwards attempts to issue a transaction t' which is trying to spend the same input(s) as t, the system will never confirm t'. If an owner issues two conflicting transactions t and t' at roughly the same time, it is possible that (a) either t or t' gets confirmed (but not both), or (b) neither t nor t' are ever confirmed. Case (b) happens if some validators see and try to confirm t, while others see and try to confirm t'. The system might stay in this state forever with the validators' approval split between t and t', with no clear majority. Crucially, such a situation can only arise if the owner of t and t' misbehaves.

The result is in every sense equivalent to the misbehaving agent losing the cryptomoney he attempted to double-spend, and does not constitute any threat to the system. The two conflicting transactions will stay present in the transaction DAG not influencing other transactions and the functioning of the system in any way, despite neither of them becoming confirmed. Agents can create new transactions (directly or indirectly) referencing both t and t', not contributing to either's approval, essentially ignoring t and t'.

It is somewhat intuitive to verify that such a system does work correctly if the cryptocurrency amounts are statically assigned to the validators, and a set of validators controlling more than two-thirds of the cryptocurrency obeys the protocol. In Section 6 we will show that our system still works correctly when the agents can freely exchange the cryptocurrency and change the appointed validators, undisturbed by the harsh conditions of an asynchronous network. Thus, we establish a system with the participation model similar to proof-of-stake protocols, but much simpler.

4 MODEL

Our blockchain is used and maintained by its participants called agents. Agents who follow the protocol are called honest. The set of agents who do not follow the protocol is controlled by the adversary. The adversary behaves in an arbitrary way. At any time, the adversary can have legitimately acquired less than one-third of the cryptocurrency present in the system, as described precisely in Section 5.4.

4.1 Communication

We assume that all agents are connected by a virtual network similar to Bitcoin's, where agents can broadcast their messages to all other agents. Like in Bitcoin, new agents can join this network to receive new and prior messages. Agents can also leave the network.

However, the network we assume is asynchronous and thus much weaker than that required by Bitcoin. The adversary controls the network, dictating when messages are delivered and in what order. Messages are only required to reach the recipient *eventually*, without any bound on the time it might take. Under such weak network requirements, an adversary delaying the delivery of messages can delay the progress of an agent, but otherwise will not be able interfere with the protocol or trick honest agents.

4.2 Cryptographic Primitives

We assume the functionality of asymmetric encryption where a public key allows every agent to verify a signature of the associated secret key. Agents can freely generate public/secret key pairs.

We also assume cryptographic hashing, where for every message a succinct, unique hash can be cheaply computed. Whenever we mention references between transactions in our protocol, we mean hashes that uniquely identify the referenced data.

Otherwise the protocol is completely deterministic.

5 PROTOCOL

In this section we describe the various components of the protocol. We refer to the cryptocurrency managed by the protocol as the money.

Outputs. Outputs are the basic unit of information. Outputs are included in transactions to identify money holders and validators. An output contains:

- *Value:* A number representing the amount of money.
- *Owner key:* A public key. The agent holding the associated secret key is the owner of the money.
- *Validator key:* A public key. The agent holding the associated secret key is indicated as the validator.

The owner controls the output. By saying that a message is signed by an output o, we will mean that the message is signed by the owner. In general, the agents could reuse their keys for multiple outputs. However, for simplicity of presentation, we do not introduce more data to identify the outputs and assume that keys always uniquely identify outputs. Any output is controlled by a single agent.

Genesis. The *genesis* is a set of outputs known upfront to every agent. Genesis describes the initial distribution of money among agents. The value of all initial outputs sums up to $3f + 1$.

5.1 Transactions

Transactions are requests issued by the agents to spend money to another agent. Outputs of transactions identify the owner A of the money, and also indicate a validator - another agent B devoted to maintaining the system, so that B can broadcast acks representing the money owner. This way, a small number of validators can contribute to transaction confirmation for the whole network. The owner A can issue a transaction to spend the money independently from B. If A wants to change the validator, A can issue a transaction spending money to itself that indicates a different validator. Of course, any agent can also indicate itself as the validator. On the other hand, the validator B can issue acks for A but cannot spend the associated money.

Every transaction is either a spending transaction or an ack. We refer to genesis as a transaction as well.

Transactions include references to other transactions. The set of transactions reachable by following references from t is called $past(t)$.

Definition 1 (spending transaction). *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 a 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 the inputs. In other words, the agent creating the transaction controls the inputs of the transaction.

Definition 2 (ack). *An ack a contains a set of references to 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 a is associated with o.*

All transactions can only reference previously created transactions with hashes. Cyclic hash references are impossible and hence all transactions form a directed acyclic graph (DAG).

5.2 Interpreting the DAG

Transactions are processed in an order respecting references. If an agent receives a transaction t but has not observed some transactions in $past(t)$ yet, the agent cannot be sure that t is in fact a transaction that does not reference invalid data. Hence, the agent does not process t until $past(t)$ is received in full. Such a situation might be compared to receiving a block without knowing the parent block in the Bitcoin blockchain.

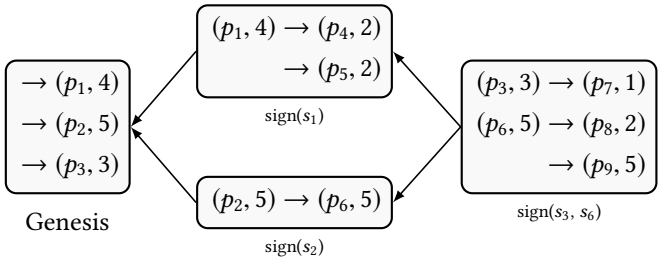

Figure 1: Example DAG of spending transactions, validator keys are omitted. The outputs of the genesis are spent in the other three transactions. The p_i's are owner keys, and s_i's are the corresponding secret keys.

Definition 3 (past). *The set of transactions reachable by following references from t is called $past(t)$. For a set of transactions T, $past(T) = \bigcup_{t \in T} past(t)$.*

Every transaction t depends on transactions that had to take place in order for t to be possible.

Definition 4 (depends). *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.*

Any two transactions that together would produce an inconsistent state of the system, such as double-spend transactions, are said to conflict.

Definition 5 (conflicts). *If two transactions u and v spend the same output, they conflict. Moreover, for two conflicting transactions u, v, every transaction that depends on u conflicts with every transaction that depends on v.*

Transactions can be confirmed by the system, and confirmation is permanent. A transaction t becomes confirmed when enough validators broadcast an *ack* to the network that indicates t as the first transaction the validator observed as spending given outputs. Intuitively, for a transaction t to become confirmed, a set of acks C_t needs to reference t. If these acks together account for more than two-thirds of the money in the system, and there is no evidence of misbehavior of the creator of t in $past(C_t)$, then t becomes confirmed. From that point on, the acks C_t serve as the proof that t is confirmed.

We now define how to interpret if an ack contributes to a confirmation of a transaction i.e. if the ack is *effective*. Genesis is confirmed from the start, effective acks are associted with confirmed outputs, confirmed transactions are indicated by effective acks, and so on.

Definition 6 (effective ack). *If an ack contributes to the confirmation of a transaction, we call it effective. An ack a associated with an 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 7 (confirmed). *Some transactions become 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 the sum of weights of acks in C_t is at least $2f + 1$.

Note that a single ack might be effective for a chain of dependent transactions, such that adding the ack to the DAG might confirm the chain at once, where there are no intermediate states with transactions being confirmed one by one.

5.3 Creating Transactions

Honest agents never attempt to spend the same output more than once. In other words, honest agents do not create conflicting transactions. Whenever an honest agent created an ack a_1, this ack will be referenced by the next ack a_2 the same honest agent creates, i.e. $a_1 \in past(a_2)$.

Algorithm 1: The protocol of issuing a new transaction t.

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

5.4 The Adversary

The adversary behaves in an arbitrary way, and thus might create conflicting transactions that do not reference each other and send them to different sets of recipients.

Any message sent by an honest agent is immediately seen by the adversary. The delivery of each message from an honest agent to an honest agent can be delayed by the adversary for an arbitrary amount of time.

Money controlled by the adversary. The value of all initial outputs sums up to $3f + 1$. We assume that the adversary controls initial outputs summing up to at most f in value. In other words, the adversary owns up to $1/3$ of the systems money at inception.

At any point the adversary can issue a transaction t sending some amount of money y to an honest agent, where the indicated validator is also an honest agent. The instant some set of confirming acks exist for t, we decrease the amount we count as owned by the adversary by y.

At any time, an honest player can issue a transaction sending some amount of money y to the adversary, or indicating the adversary as the associated validator. Then, we increase the amount we count as owned by the adversary by y.

In other words, we assume that the adversary cannot legitimately acquire (or be chosen as the validator for) more than f of the system's money.

6 DOUBLE-SPENDING

This section is devoted to proving that the presented protocol upholds ABC Consensus as defined in Section 2.

Assuming Agreement (that no conflicting transactions are ever confirmed), at least $2f + 1$ of money is always delegated to honest validators under the conditions described in Section 5.4. Hence, if there is no alternative to a transaction t, t is accepted by the system in finite time and thus All-Same-Validity and All-Same-Termination hold.

Corollary 1. *If Agreement holds, ABC protocol satisfies ABC consensus.*

We now focus on proving that if our assumptions are met, it is impossible that any two conflicting transactions are confirmed, otherwise known as the problem of double-spending.

Proof Outline. For contradiction, assume that some transaction DAG can be produced by the protocol where two conflicting transactions x and y are confirmed. Consider the instance of such G that is minimal in terms of the number of spending transactions. In Lemma 3 we show that G does not contain any unconfirmed transactions. Consider the first transaction t that becomes confirmed in G during the protocol's execution. In Lemma 4 we show that no transaction conflicting with t can become confirmed. In Lemma 6 we show that the validators of inputs of t could be replaced with the validators of outputs of t from the start of the execution. In Theorem 7 we conclude that for G to be minimal, t could be embedded in genesis with no difference to the rest of the DAG. Hence, we arrive at a contradiction with the choice of G.

Lemma 2. *If a confirmed transaction u depends on v, v is confirmed.*

Proof. Since u depends on v, $v \in past(u)$. Therefore, for any ack $a \in C_u$, $v \in past(a)$. Since dependence is transitive, v depends on a subset of transactions that u depends on. Hence, conditions of Definition 6 for v are subconditions of the definition for u. Thus, a is effective for v, and C_u is a set of effective acks for v. By Definition 7, v is confirmed. □

Lemma 3. *There are no unconfirmed transactions in G.*

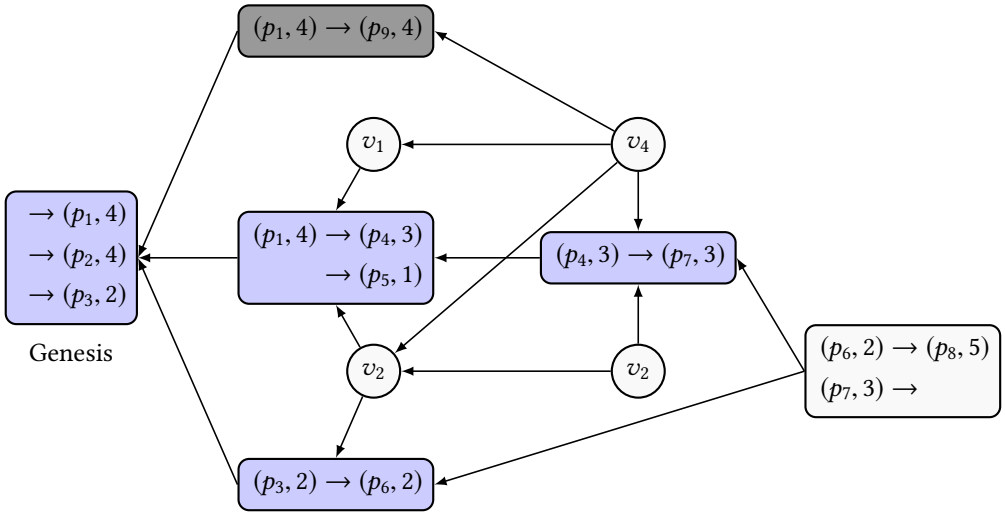

Figure 2: Example transaction DAG. Every circle node labeled v_i represents an ack sent by the validator of the output p_i. Blue transactions are confirmed based on the acks. For example, the transaction spending p_3 is confirmed by v_2 and v_4. Gray transaction t is an attempt at double-spending and will never be confirmed. The validator v_4 references t, but there is another transaction spending the same output visible in the past of the ack, hence the ack is not effective for t.

Proof. Suppose some unconfirmed transaction u exists in G. Since u is unconfirmed, by Lemma 2 no confirmed transaction depends on u.

By Definition 6 effective acks can only be issued by validators indicated in confirmed transactions. Suppose u and all dependent transactions were removed from G to obtain G'. Then, no effective ack would be removed from G. By Definition 7 any transaction confirmed in G is also confirmed in G', in particular x and y. However, G' contains less transactions than G, a contradiction with the choice of G. □

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

Proof. Since t is the first transaction confirmed during the execution of the protocol, by Definition 6 the only validators that can issue effective acks up to that point are all specified in genesis. Since the value of initial outputs for which the adversary is the validator can amount up to f, C_t has to contain honest acks of value at least $f + 1$. Let V be the validators that issued honest acks in C_t. By the protocol, V will only issue acks a such that $t \in past(a)$. Hence V cannot issue an effective ack for a transaction spending the same inputs as t. Similarly, the validators specified in t cannot issue effective acks for transactions spending the same inputs as t.

By Definition 7, there are no transactions $u \in past(C_t) \setminus \{t\}$ spending outputs associated with V.

Hence, any further transaction can obtain effective acks a such that $t \notin past(a)$ of cumulative weight at most $2f$.

Suppose for contradiction some transaction u that conflicts with t becomes confirmed. By Definition 5 and Lemma 2, there is a confirmed transaction u' spending the same input as t. By above, there cannot be effective acks for u' of cumulative weight more than $2f$, a contradiction. □

Corollary 5. *There is no transaction conflicting with t in G.*

Lemma 6. *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 genesis contains the outputs of t but does not contain the inputs of t.*

Proof. Let V_i be some validator associated with an input of t and V_o be some validator associated with an output of t. Consider some confirmed transaction u. Recall from the proof of Lemma 4 that either $u \in past(C_t)$ or $t \in past(C_u)$. Since t is spending the output of V_i, V_i can only have issued an effective ack for u if $t \notin past(C_u)$. Then $u \in past(C_t)$, and for any effective ack a issued by V_o, by Definition 6 we have $u \in past(a)$. Hence, if V_i and V_o issued an effective ack for u and v respectively, u and v cannot conflict.

Thus, t could be embedded in genesis, and the validators of the corresponding outputs can issue acks equivalent to those in G. If inputs and outputs of t do not match in value, genesis can contain smaller outputs that can combine to inputs or outputs of t in value. □

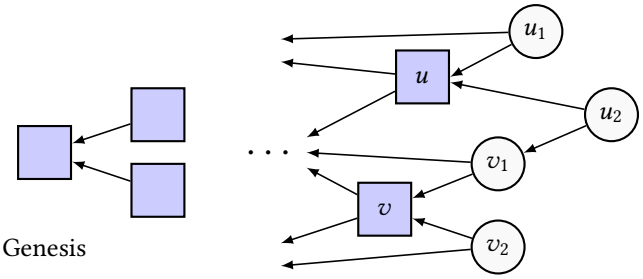

Figure 3: Example illustration of $v \in$ *past*(C_u). Transactions u_i are acks in C_u and v_i are acks in C_v. If u and v are confirmed, then $v \in$ *past*(C_u) or $u \in$ *past*(C_v).

Theorem 7. *No DAG can be produced by the ABC protocol such that a pair of confirmed transactions are conflicting.*

Proof. Suppose t is the first transaction confirmed in G during the execution of the protocol. By Corollary 5, no transaction in G conflicts with t. By Lemma 6, a DAG G' could be obtained by embedding t in genesis, with other transactions confirmed as in G, a contradiction with minimality of G. □

Corollary 8. *ABC protocol satisfies ABC Consensus.*

7 IMPROVEMENTS

7.1 Transaction Fees

To prevent spamming attacks and to incentivize maintaining the system by the validators, transaction fees can be introduced. We present a simple example fee structure, but of course many alternative schemes are possible.

As acks only serve to confirm spending transactions, they could pay no fee. Additionally, we could require acks to be effective for a certain number of new transactions in order to be valid, so that the volume of acks being broadcast is decreased and to prevent ack spamming.

Since in our setting we refrain from establishing consensus, we will also refrain from attempting to choose an agreed upon subset of validators that should receive the fee from a particular transaction. Instead, we suggest that all validators at any point are eligible to a portion of every transaction fee. For example, the issuer of an ack a of weight w can be granted additional amount ϵ to be spent as an output collected from transactions:

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

where $\mathit{effective}(a)$ is the set of transactions to which a is effective and $\mathit{fee}(u)$ is the transaction fee paid by u. The fee amount might depend on the size of the representation of u, or the number of inputs and outputs in u can be limited by the protocol.

7.2 Money Creation

If the fees are collected according to Equation (1), the fees paid by transaction issuers equal the fees collected by those confirming them, and the overall amount of money remains the same over time. However, the system might be set up so that the amount of cryptocurrency increases over time. For example, the collected fee ϵ might be multiplied by a constant $\alpha > 1$. Assuming every agent possesses less than one-third of the overall stake at any point, issuing transactions would still incur a cost as long as $\alpha \leq 3$.

As an additional role in Bitcoin and related systems, proof-of-work serves to distribute newly created money in an unbiased way. ABC could employ proof-of-work for this purpose as well. For this purpose, transactions could be allowed to include proof-of-work and receive an extra amount of stake to spend as an output. However, for these rewards to vary over time, we would need to introduce some mechanism for the protocol to record the passage of time, which we leave as outside the scope of this work.

8 RELATED WORK

Traditionally, distributed ledgers [1, 7] operate with a carefully selected committee of trusted machines. Such systems are called permissioned. The committee repeatedly decides which transactions to accept, using some form of consensus: The committee agrees on a transaction, votes on and commits that transaction, and only then moves forward to agree on a next transaction.

Bitcoin [10] radically departed from this model and became the first permissionless blockchain. In the Bitcoin system, there is no fixed committee; instead, everybody can participate. Bitcoin achieves this by using proof-of-work. Proof-of-work is a randomized process tying computational power and spent energy to the system's security, while also requiring synchronous communication. However, Bitcoin's form of consensus hardly satisfies the traditional consensus definition. Instead of terminating at any point, the extent to which the consensus is ensured raises over time, approaching but never reaching certainty. More precisely, in Bitcoin transactions are never finalized, and can be reverted with ever decreasing probability.

Similarly to Bitcoin, ABC allows permissionless participation and does not conform to the traditional definition of consensus. In contrast to Bitcoin, ABC does not rely on wasting energy solely to run the system securely, works under full asynchrony, does not rely on randomization, and prohibits reverting committed transactions.

To address the problems associated with proof-of-work, proof-of-stake has been suggested, first in a discussion on an online forum [11]. Proof-of-stake blockchains are managed by participants holding a divisible and transferable digital resource, as opposed to holding hardware and spending energy. Rigorous academic works proposing proof-of-stake systems include designs such as Ouroboros [6] or Algorand [2]. Proof-of-stake blockchains seek to solve consensus and thus rely on synchronous time. The use of (pseudo-)randomization in proof-of-stake systems is not without complications, and often considered a security risk. In contrast to proof-of-stake systems, ABC allows for completely asynchronous communication. ABC is also simpler.

In the work closest related to ours, Gupta [5] proposes a permissioned transaction system that does not rely on consensus. In this design, a static set of validators is designated to confirm transactions in a manner similar to ours.

The authors of [4] show that the consensus number of a Bitcoin-like cryptocurrency is 1, or in other words, that consensus is not needed. The paper provides an analysis and discussion of which applications rely on consensus and to what extent, all of which is directly relevant to ABC. The authors also argue that parallels can be drawn between a permissioned transaction system and the problem of reliable broadcast [8].

The authors of [9] provide an asynchronous permissioned system by relying on advanced cryptographic techniques. The main differences from ABC are that the system is permissioned, much more involved, reliant on randomization, and offers consensus.

The authors of [3] introduce a protocol based on reliable broadcast that allows participants to join and leave the system. In contrast to ABC, the protocol consists of multiple rounds of communication to agree on nodes joining or leaving the system and does not feature a functionality to delegate one's role in maintaining the system. Node communication volume increases with the number of participants, therefore it cannot be applied in permissionless contexts.

9 CONCLUSIONS

In this paper we presented ABC, an asynchronous blockchain without consensus. ABC provides the functionality of Bitcoin without consensus, without proof-of-work, without requiring synchronous communication, without relying on randomness, fast and with finality. The design of ABC is arguably the simplest possible design for a whole set of blockchain applications.

ABC provides an advantageous solution for applications like cryptocurrencies, where truthful participants do not generate conflicting status updates. However, a smart contract platform like Ethereum requires consensus to function, and it is not possible to build an equivalent system using the ABC protocol.

REFERENCES

[1] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[2] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 51–68.

[3] Rachid Guerraoui, Jovan Komatovic, and Dragos-Adrian Seredinschi. 2020. Dynamic Byzantine Reliable Broadcast [Technical Report]. *arXiv preprint arXiv:2001.06271* (2020).

[4] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. 2019. The Consensus Number of a Cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM, 307–316.

[5] Saurabh Gupta. 2016. *A Non-Consensus Based Decentralized Financial Transaction Processing Model with Support for Efficient Auditing*. Master's thesis.

[6] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*. Springer, 357–388.

[7] Leslie Lamport. 1998. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)* 16, 2 (1998), 133–169.

[8] Dahlia Malkhi, Michael Merritt, and Ohad Rodeh. 1997. Secure reliable multicast protocols in a WAN. In *Proceedings of 17th International Conference on Distributed Computing Systems*. IEEE, 87–94.

[9] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 31–42.

[10] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf. (2008).

[11] QuantumMechanic. 2011. https://bitcointalk.org/index.php?topic=27787.0.