Partitionable Blockchain

A thesis submitted to the
Kent State University Honors College
in partial fulfillment of the requirements
for University Honors

by

Joseph Oglio

May, 2020

Thesis written by

Joseph Oglio

Approved by

_____,
Advisor

_____, Chair, Department of Computer Science

Accepted by

_____, Dean, Honors College

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

The author would like to thank Dr. Mikhail Nesterenko for his support as an advisor throughout this project. The author would also like to thank Dr. Gokarna Sharma for his support, as well as Kendric Hood for his advice and guidance.

1

INTRODUCTION†

Blockchain. Peer-to-peer networks are an attractive way to organize distributed computing. Blockchain is a peer-to-peer technology for building a shared, immutable, distributed ledger. A prominent blockchain application is cryptocurrency such as Bitcoin [32] or Ethereum [42]. In a blockchain cryptocurrency, the ledger records financial transactions. The transactions are appended to the ledger block by block. This application is decentralized and peers have to agree on each block. Such consensus is the foundation of blockchain algorithms.

Classic robust distributed consensus algorithms [3, 7, 10, 19, 26, 30] use cooperative message exchange between peers to arrive at a joint decision. However, such algorithms require participants to know the identity of all peers in the network. This is often infeasible in modern high-turnover peer-to-peer networks.

Nakamoto consensus. Alternatively, Bitcoin uses Nakamoto consensus [32] where the participants compete to add blocks to the ledger. This algorithm does not require complete network knowledge. Due to the simplicity and robustness of the algorithm, Nakamoto consensus became the dominant approach to consensus in cryptocurrency design. In Nakamoto consensus, initial funds are distributed between participants. Such a participant can then submit a transaction request which moves money from one account to another.

Transactions may conflict. For example, if a transaction request is to withdraw ten dollars from an account that has only nine, this transaction is invalid. Also, if there are two requests to withdraw nine dollars from this account, then only one request can be accepted.

† This thesis is based in part on an article [21].

These requests are received by specialized peers called miners. A miner groups valid transactions into a block. To this block, the miner adds a link to the previous block. To be accepted by other miners, the newly created block has to be mined. The miners compete to get their mined blocks accepted. To resolve the competition, mining a block is made difficult.

Mining is done using a hash function. The hash function takes the data in the block and transforms it into a number, called a hash. However, in Nakamoto consensus, not every hash is acceptable. To compute an acceptable hash, a miner must add an additional number to the block. This number is called a nonce. Finding the nonce that makes the block acceptable is mining. The cryptographic hash function is designed so that it is impossible to determine the appropriate nonce except through trial and error. This process is computationally intensive. However, verifying the mined block is relatively simple: just hash the data and the nonce to check if the obtained hash is acceptable. Therefore, other miners may easily ensure that the block is properly mined.

The mined block is then broadcast to all other miners. Once a miner receives a newly mined block, it starts mining the next block by linking it to the new arrival. Thus, a chain of mined blocks is formed. This chain is called a blockchain. Since mining is performed concurrently, two blocks may be mined at the same time. This is called a fork. A fork creates ambiguity as miners cannot agree on a single block. To minimize the possibility of a fork, the mining complexity is dynamically adjusted. Bitcoin is designed to maintain the mining complexity so that the expected mining time of a block is ten minutes.

The blockchain is immutable and secure due to the algorithm design. For example, a malicious peer may try to introduce a transaction that fraudulently sends money to the account that it controls. For this, the peer needs to alter a transaction in a previously mined block. To make the block valid, the malicious peer has to re-mine it. To make this block acceptable to the network, this peer also needs to re-mine all following blocks. Since all miners in the network compete to mine blocks, the blockchain integrity is protected by the combined computing power of the network. Moreover, blockchain modification may not be easily accomplished as every miner

has its own copy of the blockchain. This distributed security is a major blockchain advantage over traditional centralized security approaches.

Partitionable blockchain. Ordinarily, a blockchain consensus algorithm assumes that the network remains connected at all times. That is, the network suspends operation while parts of the network are unable to communicate. Alternatively, a single primary partition makes progress while the others are not utilized. The problem of concurrently using all partitions is difficult since the partitioned peers can not communicate and may approve transactions that conflict and thus violate the integrity of the combined blockchain.

It is assumed that partitioning interruptions are infrequent or insignificant for network operation. However, this may not be the case. While long system-wide splits may be rare, brief separations are common. This can happen, for example, if groups of peers are connected via a small set of channels. If these channels are congested, the groups are effectively disconnected and the network is temporarily partitioned.

As blockchain becomes a larger component of the financial market, the pressure to increase the system's availability will grow. Many financial applications, such as high frequency trading [25], are sensitive to even a slight delay. A system delay that lasts a few milliseconds may cost its users substantial time and money. Hence, a blockchain-based cryptocurrency that is available through partitioning is required for this technology to realize its potential.

In this thesis, we formally address the possibility of partitionable consensus in cryptocurrency blockchain. What enables the solution is the possibility of splitting the accounts and processing transactions in the partitions independently without violating the integrity of the complete blockchain.

Blockchain improvements and related work. We use the asynchronous system model for the study of partitionable consensus. This means that a message sent to a peer can take an indefinite amount of time to arrive. A message can not be lost though so it will eventually be delivered to the peer. This model does not place

assumptions on the peers' relative computation power or message propagation delay and thus has near-universal applicability. The alternative is a synchronous system model in which all messages are delivered within a set amount of time.

Consensus is impossible in the asynchronous system even if only a single peer is allowed to crash [18]. Intuitively, peers are not able to distinguish a crashed process from a very slow one. This impossibility is circumvented with crash failure detectors [11, 13] that provide minimum synchrony to allow a solution. We pattern our investigation to this classic approach. We show the impossibility of a partitionable consensus solution in the asynchronous system and then introduce partitioning detectors to enable it.

There are a number of recent publications dealing with the implementation or modification of classic consensus [3, 7, 19, 30]. There are plenty of recent studies presenting blockchain design based on Nakamoto consensus [9, 16, 23, 24, 32, 33]. There are papers that combine classic and Nakamoto consensus [4, 34]. Recent research on Nakamoto-based blockchain often focuses on improving its speed and scalability [15, 29, 35, 44]. One promising blockchain acceleration technique is to concurrently build a decentralized acyclic graph (DAG) of blocks as opposed to a chain [36, 39]. The state-of-the-art on the blockchain consensus algorithms can be found in this recent survey [43].

The need to limit the number of forks and preserve the integrity of the blockchain limits the block confirmation rate. Since mined blocks are broadcast to the network, the block size is limited by the available bandwidth. Therefore, Nakamoto-style consensus places a limit on throughput and latency. Presently, Bitcoin confirms about 7 transactions per second [28]. For comparison, VISA may process up to 24,000 transactions per second. Likewise, Bitcoin's latency compares unfavorably with other currency technologies [1].

Bitcoin is also a major source of power consumption. In order to try billions of nonces, store the roughly 250 GB, and transmit all the messages needed for the system the network used nearly the same amount of electricity as Ireland [41]. That was in 2014 since then the power consumption has only increased. These issues are why Bitcoin lacks scalability, this means that by adding peers to the network,

the network does not improve. As the mining time is always roughly ten minutes adding a peer only increases communication and electricity overhead.

To combat these issues and others new blockchain algorithms have changed the longest chain rule, the proof-of-work algorithm, and added various other tools [9, 23, 24, 33, 36, 42]. Even with these changes the majority of them still follow the same basic rules needed in this thesis. This means even as blockchain gets more sophisticated our contribution will still be relevant.

Relatively few publications focus on partitionable blockchains. There are some studies on the partitionable classic consensus [5] using failure detectors. Partitionable consensus has similarities with the group membership problem, which deals with presenting a consistent membership set to the processes despite process and link failures [8, 12, 14, 17, 37, 38].

Extended virtual synchrony (EVS) [31] is a technique that supports continued operation in all partitions. That is, during network partitioning and re-connection, it maintains a consistent relationship between the delivery of messages and configuration changes. However, static membership is assumed and hence this is not easily extendable to work in blockchain systems with dynamic membership.

Tran *et al.* [40] consider an algorithm that implements partitionable blockchain consensus in the context of swarm robotics. In swarm robotics, the robot swarms may experience network partitions due to navigational and communication challenges or in order to perform certain tasks efficiently. Their solution extends EVS and hence is not suitable for partitionable blockchain under dynamic membership as we consider in this thesis. Recently, Guo *et al.* [20] noticed that synchronous classic consensus protocols cannot even tolerate a short-term jitter that makes a node offline (or leave the system) for a very short time. Then they provided a solution that makes those protocols tolerant to such jitters, keeping the same consistency and liveness properties.

Karlsson *et al.* propose a partitionable blockchain for Internet-of-things devices [22].

Contribution and outline. The rest of the thesis is organized as follows. We present

the partitionable blockchain in Chapter 2. Specifically, we present notation used throughout in Section 2.1. In Section 2.2, we formally define the Partitionable Blockchain Consensus Problem. To the best of our knowledge, this problem has been defined for the first time. In Section 2.3, we prove that this problem is not solvable in the purely asynchronous system.

In Section 2.4, we define detectors that enable a solution to the problem and explore the relationship between the detectors. The key detector determines the block age: whether the block was mined before or after the network is split into partitions. A perfect block age detector *AGE* does not make mistakes. An eventual block age detector $<AGE$ makes finitely many mistakes. In Section 2.5, we present an algorithm *PART* and prove that it solves the Partitionable Blockchain Consensus Problem when it is combined with *AGE*, We then extend the proof to apply to $<AGE$ and an even weaker detector *WAGE*. We evaluate the performance of *PART* in Section 2.6. We extend this solution to handle multiple network splits in Section 2.7. We conclude the thesis with further algorithm extensions and possible future work in Chapter 3.

2

PARTITIONABLE BLOCKCHAIN

2.1 notation

Communication model. A *peer* is a single process. All peers operate correctly and do not have faults. A *partition* is a collection of peers that can communicate. This communication is done through message passing. A broadcast sends a message to every peer in its partition. Communication channels have infinite capacity and are FIFO. The channels are reliable unless the network is split. A *(network) split* is an event that separates one partition into two. A message sent before the network split is always delivered. A message sent after the split is delivered only to the recipients that are in the same partition as the sender.

Each peer contains a set of variables and commands. A *network state* is an assignment of a value to each variable from its domain. A command transitions the network from one state to another. Command executions are atomic and do not overlap. A *computation* is a sequence of such states which is either infinite or ends in a state where no command may be executed.

We assume fair message receipt and fair command execution. Specifically, in any computation, a sent message is eventually received and a command is either executed or disabled.

We assume there is at most one split per computation. That is, the network starts as one partition and it may split into two. There are no re-connections. The two partitions exist for the rest of the computation.

Accounts and transactions. An *account* is a means of storing funds. Each account has a unique identifier. A *transaction* is a transfer of funds from the source to the target account. We assume that there is a single source and a single target account.

Each transaction has a unique identifier as well. An *account balance* is the total amount of funds that were transferred to the account minus all the funds that were transferred out.

A *client* is an entity that submits transactions to the network, we assume all clients are honest. A transaction is submitted by broadcasting it. There may be multiple clients. The transaction identifiers for each client are monotonically increasing. A client submits each transaction to a single partition. If transactions are submitted to two partitions, they are considered separate transactions.

Blockchain. Peers mine transactions. Such a mined transaction is a *block*. That is, we assume a single transaction per block. A mined block cannot be altered. Besides a transaction, each block contains an identifier of another block. Thus, a block is linked to another block. A *blockchain* is a collection of such linked blocks. A *genesis* is the first block in the blockchain. The genesis is unique. There are no cycles in the blockchain. That is, the blockchain is a tree. A *branch* of a tree is a chain of blocks from the genesis to one of the leaves of the tree. See, for example, a branch from the genesis to block 1 in Figure 2.3.

The *main chain* is the longest branch in a blockchain. Ties are broken deterministically. *Permanent branch* is infinite. We assume that there is at most one permanent branch per partition.

Each peer operates as follows. If it receives a transaction, it stores it. The peer attempts to mine one of the pending transactions by linking it to the tail of its main chain. If it succeeds, the block is broadcast. The peer continues while there are pending transactions. We make the following assumption.

Assumption 1. *A peer either receives infinitely many blocks or mines infinitely many blocks.*

Global blockchain (tree) is the collection of all blocks mined by all peers. A *fork* is a pair of blocks that link to the same block. A fork happens when two peers succeed in concurrently mining blocks. See Figure 2.3 for an example. Since the genesis block is unique, it may never be in a fork. If there is a network split, a *seed* is the last block mined on each branch before the split.

A transaction is *valid* if its application leaves the source account with a non-negative balance. The transaction is *invalid* otherwise. An account balance is determined by the sequence of transactions on a particular branch. Therefore, a transaction may be valid in one branch and invalid in another. If a peer mines a transaction, it is valid relative to its main chain.

A transaction is *confirmed* if it is in the permanent branch. It is *rejected* if it is not in the permanent branch. A transaction is *resolved* if it is either confirmed or rejected.

A transaction is *permanently valid* if it is valid indefinitely or until it is resolved. We assume that in each partition, at least one client submits infinitely many permanently valid transactions. A transaction is *permanently invalid* if it is invalid indefinitely or until it is resolved.

We assume that the same transaction may not exist in two separate partitions. This means that in case of a split, the validity of a transaction may be maintained in one partition only. In other words, a pre-split transaction may not be valid in both partitions.

In case of a network split, the account balances are also split. We assume that at least some funds are distributed between partitions. That is, we exclude the case where a partition is left with zero funds for all accounts. Otherwise we place no restrictions on the way that accounts are split between partitions so long as the total on the account balance in both partitions post-split is equal to the pre-split account balance in the seed block. For example, in the network there is an account a that has a balance of 100. Then a network split occurs. Account a may be split into a_1 and a_2. Accounts a_1 and a_2 cannot be in the same partition. Account a is split 70/30, thus the balance of a_1 is 70 and a_2 is 30. If the accounts are split unevenly, each peer must know to which partition it belongs. If a is halved post-split, then the peers do not need to identify which partition they are in.

A *branch merge* is an arbitrary interleaving of transactions of two branches such that the order of transactions of each branch is preserved. Two branches are *mergeable* if any branch merge retains the validity of all transactions of the two branches.

A *seed* is the block where the accounts are split. Observe that in a single computation, accounts may potentially be split on different seeds.

Proposition 1. *Branches from different partitions are mergeable if they are split on the same seed.*

Detectors. A *detector* is a mechanism that provides information to the algorithm that it may not able to determine otherwise. The *pure asynchronous* system has no detectors.

By its outputs, a detector produces a computation. Detector A is *weaker* than detector B, if there exists an algorithm such that it accepts every computation of A and produces a computation of B. Detector A is *equivalent* to B, denoted $B \equiv A$, if both A is weaker than B and B is weaker than A. Detector A is *strictly weaker* than B, denoted $B >\!\!- A$, if A is weaker than B but not equivalent to B.

2.2 the partitionable blockchain consensus problem

Definition 1. The Partitionable Blockchain Consensus Problem is the intersection of the following three properties:

confirmation validity: no invalid transaction is confirmed;

branch compatibility: permanent branches are mergeable;

progress: every permanently valid transaction is eventually confirmed.

The first two properties are safety while the progress property is liveness [6].

2.3 impossibility

We show that it is not possible to achieve partitionable blockchain consensus in the asynchronous system. Intuitively, this is due to the lack of information whether a particular message reaches all peers during the network split. Recall that our model

guarantees reliable pre-split message delivery while after split this guarantee is only within the sender's partition. Thus, a sender may not be sure whether all the peers received its message. This uncertainty is exploited in the following theorem.

Theorem 1. *It is impossible to solve the Partitionable Blockchain Consensus Problem in the pure asynchronous system.*

Proof. Assume that there is an algorithm *ALG* that solves the Partitionable Blockchain Consensus Problem in the pure synchronous system.

Let two transactions t_1 and t_2 be such that they are valid unless one is confirmed. The confirmed transaction invalidates the other. Observe that if these two transactions are submitted, then *ALG* may only accept one and only one of these transaction. Indeed, if *ALG* accepts both t_1 and t_2, it violates the confirmation validity property since it accepts an invalid transaction. Similarly, if it rejects both t_1 and t_2, it violates progress since it rejects two permanently valid transactions.

Consider computation c_1 which contains a split. Let p_1 and p_2 be two resultant partitions. Each partition has a permanent branch. Let blocks b_1 and b_2 be two mined blocks containing t_1 and t_2, respectively.

In c_1, peers of p_1 receive b_1 while peers in p_2 receive b_1 and b_2. Since, in the absence of b_2, b_1 contains a permanently valid transaction, peers in p_1 confirm b_1. For branch compatibility, peers in p_2 must also confirm b_1. This means that they have to reject b_2.

Consider computation c_2 which has the same partitions p_1 and p_2 as in c_1. However, the peers in both partitions only receive b_2. Due to the progress property, both partitions are bound to confirm this block.

Let us compose a computation c_3 with the same partitions, where peers in p_1 receive b_2 while peers in p_2 receive both b_1 and b_2. We compose the computation such that the behavior of peers in p_1 is as in c_2 and in p_2 as in c_1. That is, the peers in p_1 confirm b_2 while peers in p_2 confirm b_1 and reject b_2. However, this means in c_3 one partition confirms a transaction while the other partition rejects it. That is, this computation of *ALG* violates the branch compatibility property of the Partitionable Blockchain Consensus Problem. Hence, contrary to our assumption, *ALG* may not be a solution to this problem. □

2.4 partitioning detectors

The partitionable blockchain consensus problem is impossible without detectors. The lack of solution is due to the impossibility of ascertaining whether the message reached all peers. Let us discuss detectors that may circumvent this and enable a solution. A propagation detector *PROP* addresses this concern directly. Specifically, for each peer, *PROP* determines whether a particular block is delivered to peers of the entire network or just for a single partition. Note that due to the potentially long message delay in the asynchronous system, the output of such a detector may also be delayed.

Another detector *AGE* determines whether the block was mined before or after the split. The detector classifies the pre-split block as *old*, and post-split block as *new*. Since the block is broadcast right after mining and message transmission is reliable, the only way for a message not to be delivered to all peers is if there is a split in the network. Hence the following lemma.

Lemma 1. *The propagation detector is equivalent to the age detector. That is:* $PROP \equiv AGE$.

Detector eventual *AGE*, denoted $<AGE$, is similar to *AGE*. Like *AGE*, detector $<AGE$ outputs whether the block was mined before or after the split. However, $<AGE$ is not reliable. Specifically, $<AGE$ may make a finite number of mistakes. To put another way, for each block, $<AGE$ is guaranteed to eventually output a correct result. We call *AGE* perfect age detector to distinguish it from eventual *AGE*.

Lemma 2. *The perfect AGE detector is strictly stronger than eventual AGE. That is:* $AGE >- <AGE$.

Proof. To prove strict weakness of $<AGE$, we show that there does not exist an algorithm that takes any computation of $<AGE$ and produces a computation of *AGE*.

Assume the opposite. Let *ALG* be such an algorithm. Let computation c_1 of $<AGE$ decide the age of a single block b. The block is old. Perfect *AGE* does not make

mistakes. Therefore *ALG* has to output that the block b is old, on the basis of the output of $<AGE$. Let s_1 be the state of c_1 where *ALG* outputs the decision of *AGE*.

We compose the computation c_2 as follows. It contains the same block b and the same output of $<AGE$ up to and including s_1. However, in this computation b is new and $<AGE$ is mistaken, $<AGE$ corrects its mistake later in the computation of c_2. However, since c_1 and c_2 share prefixes, *ALG* outputs that b is old. That is, *ALG* makes a mistake. Therefore, *ALG* may not be an implementation of *AGE*. This means that our initial assumption is incorrect and the lemma follows. □

Detector *WAGE*, pronounced "weak-age", has output similar to *AGE* and $<AGE$. However, it may make infinitely many mistakes subject to the following constraints. For each block (i) at least one peer per partition makes only finitely many mistakes; (ii) every peer outputs correct results infinitely often. To put another way, *WAGE* ensures that at least one peer eventually starts classifying blocks correctly and all other peers at least alternate their classifications without permanently settling on incorrect classifications.

Lemma 3. *Eventual age detector is equivalent to weak age detector. That is:* $<AGE \equiv$ WAGE.

Proof. All computations of $<AGE$ are already computations of *WAGE*. To prove the equivalency, we need to show that $<AGE$ can be implemented using *WAGE*. We discuss the implementation of the detector for a single block. For multiple blocks, the detector implementation runs concurrently.

The implementation algorithm is shown in Figure 2.1. It operates as follows. Each peer p, maintains the last known output of *WAGE* for all peers. It is stored in array *ages* indexed by peer identifier. Similarly, p keeps track of the number of changes in the output of *WAGE* for each peer. This is stored in array $flips$. If the output of *WAGE* changes for its peer, it updates $ages[p]$, increments $flips[p]$ and broadcasts the update. For implemented output of $<AGE$, each peer outputs the value of *WAGE* with the minimum number of flips.

Let us discuss why this implementation is correct. Let p be a peer that makes finitely many mistakes in a computation. In this computation, $flip[p]$ is finite in all

constants
p // identifier of this peer
b // block whose age is evaluated

variables
$flips$ // array of changes in output from each peer, initially zero
$ages$ //array of most recent outputs of $WAGE$, initially old

commands
$ages[p]$ **not** $= WAGE(b) \longrightarrow$
$ages[p] = WAGE(b)$
increment $flips[p]$
$broadcast(flips[p], ages[p])$

receive $numFlips, age$ **from** $id \longrightarrow$
$flips[id] := numFlips$
$ages[id] := age$

$<AGE(b) \longrightarrow$ // implemented output of $<AGE$
output $ages[id]$ for the id with minimum $flips[id]$

Figure 2.1: Implementation of $<AGE$ using $WAGE$.

peers. If p makes infinitely many mistakes, $flip[p]$ grows without a bound in all peers. By the specification of *WAGE*, for each block b, there is at least one process per partition that makes finitely many mistakes. Our implementation selects the output of implemented $<AGE(b)$ to be the one with the smallest number of flips. This way, in any computation, eventually, the output of $<AGE(b)$ is correct. □

Detector *SPLIT* outputs whether split in the system has occurred. Observe that the message delivery is unreliable only in case of a network split. Therefore, the occurrence of the split can be determined if *PROP* indicates that a certain block has not propagated to the whole system. The converse is not in general true. Just the fact of a split does not allow peers to determine whether the particular block has reached every peer. Hence the following lemma.

Lemma 4. *The propagation detector is strictly stronger than the split detector. That is* $PROP >\!\!- SPLIT$.

This theorem summarizes the above lemmas.

Theorem 2. *The relationship between partitioning detectors is as follows:*

$$PROP \equiv AGE >\!\!- <AGE \equiv WAGE,\ PROP >\!\!- SPLIT.$$

2.5 algorithm part

In this section we present an algorithm, we call *PART*, that solves the Partitionable Blockchain Consensus Problem. This algorithm is shown in Figure 2.2.

Algorithm description. Each peer maintains its copy of the blockchain *bc*, and a priority queue *txs* of all received transactions. The transactions are arranged in the order of their identifiers in *txs*. If a new transaction is received, it is entered into *txs*. We assume that *txs* is never empty. See Figure 2.3 for an example of a blockchain.

PART keeps track of the most recent output of *AGE* detector in the *currentAge* variable. When *currentAge* is *new*, the block is mined with split accounts. The value

of *currentAge* is recorded in the mined block. Once the block is mined *PART* checks the output of *AGE* against the new block and sets *currentAge* accordingly.

Function *mainChain*() of *PART* operates as follows. It consults *AGE* for all blocks in *bc* and constructs *trueTree* with only those blocks whose recorded age matches the output of *AGE*. If *AGE* is perfect, the *trueTree* contains all of *bc*. Function *mainChain*() then builds *oldTree* that contains only old blocks that are connected to the genesis block. Function *mainChain*() then finds the longest branch *oldBrach* in *oldTree*. Ties are broken deterministically. Then, *mainChain*() examines the new branches of *trueTree* connected to the tail of *oldBranch* and selects the longest branch which it stores in *newBranch*. Function *mainChain*() returns the concatenation of *oldBranch* and *newBrach*, To summarize, *mainChain*() operates on the subtree whose age agrees with the output *AGE* and returns the longest old branch connected to the longest new branch. We refer to the output of *mainChain*() as *main chain*.

Let us discuss the operation of *mainChain*() using the example in Figure 2.3. The *oldTree* there includes all blocks in the branches that run from the genesis to seeds *A*, *B*, *C* and *D*. Branches that run to *C* or *D* are of equal length and longer than the others. Assume the tie is broken in favor of *C*. Two new branches are attached to *C*. The branch that runs to block 2 is selected since it is the longer one. Function *mainChain*() returns the branch that runs from the genesis to block 2. Note that even though the branch that runs to 1 is longer overall, old blocks are considered first. Thus, branch to 1 has a shorter old blocks branch.

Function *nextValid*() returns the fist valid transaction in *txs* that is not in the main chain. Function *resetMining*() checks whether the peer is currently mining the next valid transaction, and if not, it restarts the mining.

PART operates as follows. Each peer continuously attempts to mine the first valid transaction in *txs* that is not in the main chain of *bc*. Each peer tries to mine the first transaction that is not already included in its main chain. If mined, the recorded age of the block is compared to the output of the *AGE* detector. If they are the same, the newly mined block is added to *bc* and broadcast. If not, new age is recorded in *currentAge*, and the block is discarded. Then, the mining of the next transaction starts.

If the peer receives a block *nb* mined by another peer, it checks if this block is linked to any of the blocks in *bc*. If not then this block is added to *unlinked*, which is a set of such blocks. If *nb* is linked to *bc*, the peer inserts this block into *bc* then checks if any of blocks in *unlinked* may now also be linked to *bc*.

Correctness proof.

Lemma 5. *The main chain of every peer increases indefinitely.*

Proof. Let us consider the position of each peer on the global blockchain tree according to the peer's main chain. The peer may change its position by receiving a mined block or by mining a block itself. Let us consider the position change due to receiving a block. If the peer receives a block linked to its current branch, it moves up the branch and extends its main chain. If the peer receives a block linked to a different branch, it may move to the new branch. Let us examine this move. Let *cb* be the current branch and *nb* be the new branch.

Each branch is a concatenation of two chains: a prefix of old blocks and a suffix of new blocks. The move happens if the prefix of *nb* is longer than the prefix of *cb*; or the prefixes are the same and the suffix of *nb* is longer than the suffix of *cb*.

If the computation contains no split, then every block that is mined is an old block. In this case, the new suffixes do not exist and the peer moves only if the prefix of *nb* is longer.

If the computation contains a split, the peer may potentially move to a shorter new branch because it has a longer prefix. However, if there is a split, the number of old blocks is finite. Once every peer receives all the old blocks, such moves are not longer possible. That is, the number of times the peer switches to a shorter branch is finite. After these switches, the peer may switch branches only if it has a longer suffix. That is, if this new branch is longer. Thus, if the peer switches position due to block receipt infinitely many times, its main chain grows indefinitely.

Let us consider a peer *p*, that changes position due to block receipt only finitely many times. This means, by Assumption 1, that *p* mines infinitely many transactions itself. In which case its main chain grows indefinitely as well. □

Lemma 6. *PART satisfies the progress property of the Partitionable Blockchain Consensus Problem.*

Proof. Let t be a permanently valid transaction. Let us consider the suffix of the computation where each peer p receives t. Once p receives t, it enters t into txs. Each client submits transactions in the increasing order of their identifiers. That is, in any computation, there is only finitely many transactions with identifiers smaller than t.

By Lemma 5, the main chain of every peer increases indefinitely. Every block in this main chain is mined by some peer. Hence, there must be at least one peer p that mines infinitely many blocks in this main chain. By the design of the algorithm, each peer mines the valid transaction with the smallest identifier that is not in its main chain. Therefore, p may only mine finitely many blocks before t. Thus, an infinite main chain must contain t. □

Lemma 7. *In PART, the permanent branches of all peers have a common prefix up to and including the seed block.*

Proof. Since the message propagation is reliable, blocks mined before split will be sent and received by all peers of the network. *PART* computes the main chain to be the branch that is the longest distance from the genesis to the seed. The ties are deterministically broken. Therefore, once all peers receive all pre-split messages, their main chains include the same branch and the same seed. □

Lemma 8. *Algorithm PART satisfies the branch compatibility property of the Partitionable Blockchain Consensus Problem.*

Proof. According to Lemma 7, the main chains of all peers include the same seed block. Due to Proposition 1, branches from different partitions are mergeable. Hence the lemma. □

Theorem 3. *Algorithm PART solves the Partitionable Blockchain Consensus Problem.*

Proof. *PART* never confirms an invalid transaction. Thus, *PART* satisfies confirmation validity. Branch compatibility satisfaction is shown in Lemma 8. Progress satisfaction by *PART* is shown in Lemma 6. □

Observe that the presented algorithm operates correctly even if the detector makes finitely many mistakes. That is, if the detector is $<AGE$. We call this combination $PART + <AGE$. Indeed, once $<AGE$ converges to the correct output for all blocks and each peer receives all pre-split blocks, all peers agree on the seed. From this point, *PART* operates correctly.

Per Theorem 2, $<AGE$ may be implemented using a weaker detector *WAGE*. Let $PART + WAGE$ be the combination of *PART* and such an implementation. Such a combination also solves the partitioning problem. Hence the following theorem.

Theorem 4. *Algorithm* $PART + <AGE$ *and* $PART + WAGE$ *solve the Partitioning Blockchain Consensus Problem.*

2.6 performance evaluation

Setup. We evaluate the performance of *PART* using an abstract simulation. We study the behavior of our algorithm through computations that we construct. The code for our simulation is available on GitHub [2].

The simulated network consists of n peers. An individual computation is a sequence of rounds. In every round, each peer may receive a single new message from each other peer, do local computation, and send messages to other peers.

Message propagation may take several rounds. Each message is delayed by a number of rounds. The number of rounds a message is delayed is selected uniformly at random. That values range from 1 to maximum delay d. Concurrent messages in the same channel are delayed further. That is, given a peer p_s that sends several concurrent messages to the same peer p_r, each message is impeded by a random delay plus the sum of the delays of all messages sent before it in that channel. Each peer in the network has a unique channel to every other peer. Message delivery is FIFO. In a single round, a peer may only receive a single message from each sender.

The transaction submission rate is constant: one transaction per round. A submitted transaction is broadcast by a randomly selected peer.

Block mining is simulated. Mining time is as follows. Each peer has an oracle that tells the peer whether it mined a block. In every round, the probability of mining a block for each peer is uniformly distributed between 1 and $d \cdot n$.

The network size is 100 peers. A split separates the network into two equally sized partitions of 50 peers. Overall transaction rate or mining rate does not change in the event of a split.

We measure algorithm throughput. We define it as follows: the ratio of confirmed to submitted transactions. We observe the change in throughput as the computation progresses. We run 1000 experiments per data point.

Results and analysis. The results of our experiments are shown in Figures 2.4, 2.5, and 2.6. In all figures, we plot the throughput achieved up to a particular round in a computation. The figure is averaged across all computations. We measure the throughput for maximum network delay of $1, 2, 3$, and 10 rounds.

Figure 2.4 shows the results of the experiments with *PART* and a perfect *AGE* detector. That is, the detector never makes mistakes. The split occurs at round 100 of the computation. At first, the throughput increases until it saturates: the number of confirmed transactions matches the number of newly submitted ones. Throughput is lower with higher network delay since it takes longer for transactions and blocks to propagate.

After the split, there are two smaller size partitions. The probability of a fork in each partition decreases. Which means that, even though the transaction rate and mining rate does not change with split, the transaction throughput increases.

In Figure 2.5, we show the results of the experiments with *PART* and $<$*AGE*. The split occurs in round 100. However, $<$*AGE* continues to classify all blocks as old until round 200. The detector operates correctly afterwards. Between rounds 100 and 200, while the detector classifies blocks incorrectly, transactions are not confirmed. Therefore, the throughput decreases. Once the detector corrects itself, the old blocks are ignored by the algorithm, new blocks are mined and the throughput recovers, This post-split throughput is expected to reach the post-split throughput of the perfect split detector. For example, the throughput for delay 1 would approach 0.7.

Figure 2.6 also shows the results of *PART* with $<$*AGE*. In this case, there is no split, however, between rounds 100 and 200, the detector classifies all blocks as new. The detector recovers and starts classifying all blocks as old after round 200. The algorithms behavior is similar to that shown in Figure 2.5.

Note, that there is no actual split in this experiment. Therefore, after the detector recovers, the number of forks does not decrease. Hence, the throughput is lower than that shown in Figure 2.5. The throughput is expected to reach pre-split throughput of the perfect split detector. For example, the throughput for delay 1 would approach 0.6.

Our simulation results demonstrate the performance of *PART* under different parameters. They should serve as a reference for the algorithm and detector implementation.

2.7 multiple splits

Let us consider the case of multiple split events in a single computation. That is, a network partition may further split. Each individual split event separates a partition into two. We introduce two new detectors to handle this case.

The perfect multiple block age detector *MAGE* is the modification of *AGE*. *MAGE* operates as follows. For each block b, *MAGE* outputs the number of split events that happened in the partition where this block is mined. For example, if the partition is never split, *MAGE* outputs 0. If the partition split into two A and A^t, *MAGE* outputs 1 for the peers of both partitions. If A splits into B and B^t, then *MAGE* outputs 2 for the peers in B and B^t and still 1 for the peers of A^t.

The eventual multiple block age detector $<$*MAGE* and weak multiple block age detectors *WMAGE* are defined similarly to their single-split counterparts.

Algorithm *PART* operates with *MAGE*, $<$*MAGE* and *WMAGE* without modifications. We summarize this in the following theorem.

Theorem 5. *Algorithms PART + MAGE, PART +* $<$*MAGE, and PART + WMAGE solve the Partitionable Blockchain Consensus Problem with multiple splits.*

```
variables
    bc    // tree of mined blocks, rooted in genesis
    unlinked    // set of received blocks with missing intermediate
            // links
    txs    // queue of received transactions, prioritized by id
    currentAge  // true if accounts are split after partitioning

functions
    mainChain()
        trueTree := blocks in bc whose age match AGE output
        oldTree := branches with only old blocks of
                        trueTree rooted in genesis
        oldBranch := longest branch in oldTree
        newTree := branches with only new blocks of
                        trueTree rooted in tail(oldBranch)
        newBranch := longest branch in newTree
        return oldBranch + newBranch

    nextValid() // returns the first valid
                    // transaction in txs that is not in main(bc)

    resetMining()
        if not mining nextValid()
            startMining nextValid() with currentAge

commands
receive transaction t -->
    enqueue(txs, t)
    resetMining()

mine block nb -->
    currentAge := AGE(nb)
    insert(nb, bc)
    broadcast(nb)
    resetMining()

receive mined block nb -->
    if possible to insert(nb,bc)    // add new block to blockchain
        insert(nb, bc)
        while exists b in unlinked that can be inserted into bc
            insert(b, bc)
    else
        add nb to unlinked
    resetMining()
```

Figure 2.2: Algorithm *PART*.

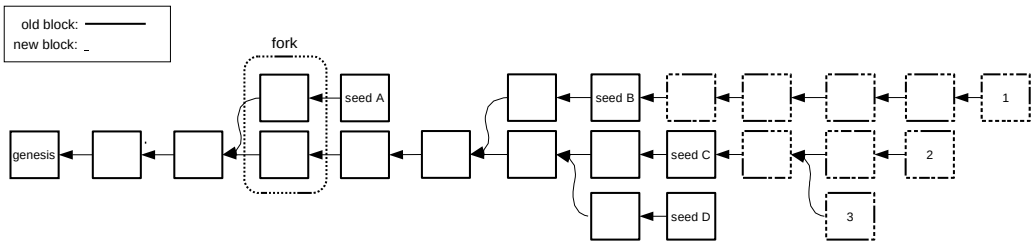

Figure 2.3: Global blockchain tree generated by *PART*.

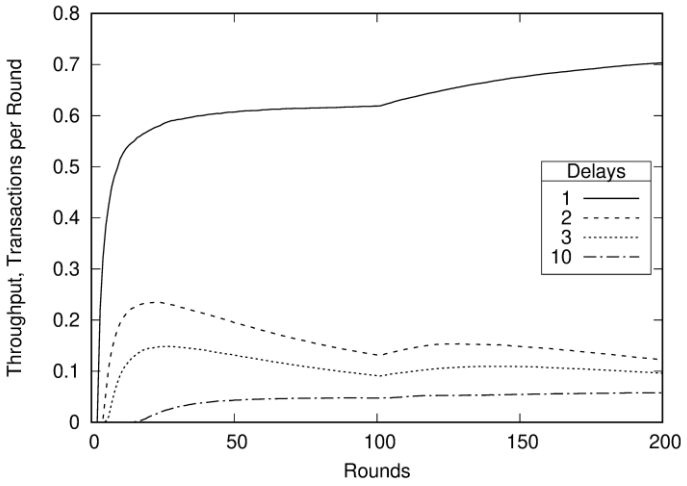

Figure 2.4: *PART* with perfect *AGE*. Split at round 100.

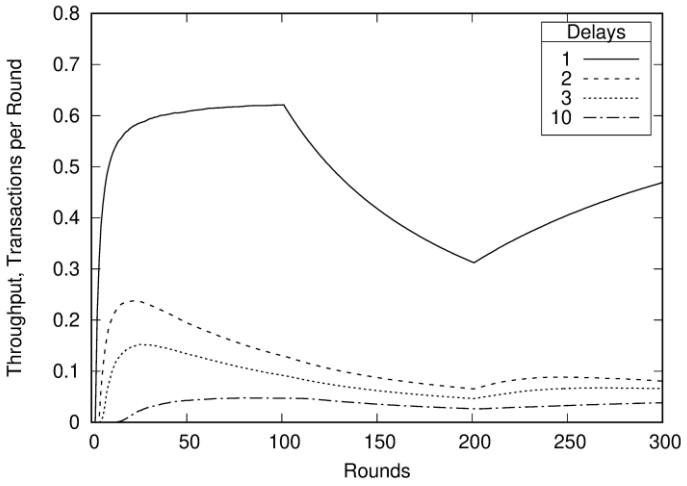

Figure 2.5: *PART*+<*AGE*. Split at round 100, detector recognizes it at round 200.

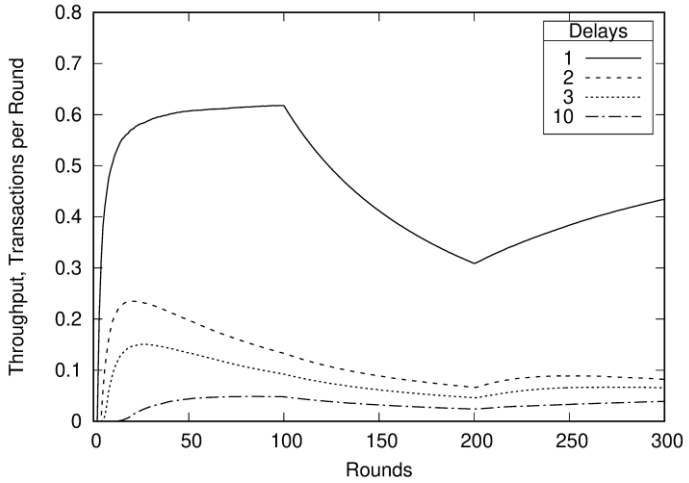

Figure 2.6: *PART+<AGE*. No split in computation, detector mistakenly recognizes it at round 100, corrects at round 200.

3

CONCLUSION, EXTENSIONS AND FUTURE WORK

In this thesis, we stated the Partitionable Blockchain Consensus problem, analyzed and provided a detector-based solution to it. Our approach combines theoretical rigor with a practical problem of importance to the industry. We believe that this is an interesting contribution to the field. Our solution may be further extended. We outline some of the extensions and future work below.

Partition merging, detector implementation, block purging. The pure asynchronous system allows us to reason about the essential properties of the algorithm that do not rely on timing assumptions. Nonetheless, we would like to discuss the practical time-based implementation concerns of our algorithm.

Partition merging may not be studied in an asynchronous system: the split is either permanent or it does not occur at all. Indeed, in the asynchronous system, a temporary loss of connectivity is a special case of message delay. However, in practical systems, partitions operating independently may need to be reconciled once the network connectivity is restored. In this case the operation is as follows: the partitions exchange their trees, select the longest tip in each sub-tree, make a seed block out of these two tips and proceed mining on this combined seed. The account balances are also merged by reversing the operation of account balance splitting at network partition.

The age detector may be implemented with checkpointing. The idea is as follows. The peers agree on a checkpoint block on every branch of the blockchain. Once the split occurs, if a certain block precedes the checkpoint block, it is considered old. A block is new if it follows the checkpoint block. To limit the rollback overhead, the checkpoints are moved closer to the leaves of the blockchain as the computation progresses.

Checkpoints can also be utilized to save the memory used to store the blockchain. Since the peers never roll back past checkpoint blocks, rather than storing individual old blocks, it is sufficient to just store the resultant checkpoint account balances. The old blocks may then be purged from memory.

Fault tolerance. Let us discuss how the proposed partitionable blockchain may withstand other faults. Peer crashes may be problematic as the blockchain has no way of determining whether the split partition is operational or it crashed. In this case, the crashed partition leads to the loss of its share of account balances. To enable crash tolerance, a crash failure detector [11, 13] may need to be incorporated in the design.

A robust blockchain needs to be tolerant to Byzantine faults [27] where affected peers may behave arbitrarily. Byzantine peers may compromise agreement on the seed block or on the split itself. We believe that our proposed algorithm may be made tolerant to such faults. However, a definitive study is left for future research.

BIBLIOGRAPHY

[1] https://usa.visa.com/run-your-business/small-business-tools/retail.html).

[2] https://github.com/khood5/distributed-consensus-abstract-simulator.git.

[3] Ittai Abraham, Srinivas Devadas, Kartik Nayak, and Ling Ren. "Brief Announcement: Practical Synchronous Byzantine Consensus." In: *DISC*. 2017, 41:1–41:4. doi: 10.4230/LIPIcs.DISC.2017.41. url: https://doi.org/10.4230/LIPIcs.DISC.2017.41.

[4] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. "Solida: A blockchain protocol based on reconfigurable byzantine consensus." In: *arXiv preprint arXiv:1612.02916* (2016).

[5] Marcos Kawazoe Aguilera, Wei Chen, and Sam Toueg. "Using the Heartbeat Failure Detector for Quiescent Reliable Communication and Consensus in Partitionable Networks." In: *Theor. Comput. Sci.* 220.1 (1999), pp. 3–30. issn: 0304-3975. doi: 10.1016/S0304-3975(98)00235-7. url: https://doi.org/10.1016/S0304-3975(98)00235-7.

[6] Bowen Alpern and Fred B Schneider. "Defining liveness." In: *Information processing letters* 21.4 (1985), pp. 181–185.

[7] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains." In: *Proceedings of the Thirteenth EuroSys Conference*. 2018, pp. 1–15.

[8] Özalp Babaoğlu, Renzo Davoli, and Alberto Montresor. "Group membership and view synchrony in partitionable asynchronous distributed systems: Specifications." In: *ACM SIGOPS Operating Systems Review* 31.2 (1997), pp. 11–22.

[9] Iddo Bentov, Rafael Pass, and Elaine Shi. "Snow White: Provably Secure Proofs of Stake." In: *IACR Cryptology ePrint Archive* 2016.919 (2016).

[10] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance and Proactive Recovery." In: *ACM Trans. Comput. Syst.* 20.4 (Nov. 2002), pp. 398–461. issn: 0734-2071. doi: 10.1145/571637.571640. url: http://doi.acm.org/10.1145/571637.571640.

[11] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. "The weakest failure detector for solving consensus." In: *Journal of the ACM (JACM)* 43.4 (1996), pp. 685–722.

[12] Tushar Deepak Chandra, Vassos Hadzilacos, Sam Toueg, and Bernadette Charron-Bost. "On the impossibility of group membership." In: *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*. 1996, pp. 322–330.

[13] Tushar Deepak Chandra and Sam Toueg. "Unreliable failure detectors for reliable distributed systems." In: *Journal of the ACM (JACM)* 43.2 (1996), pp. 225–267.

[14] Gregory V Chockler, Idit Keidar, and Roman Vitenberg. "Group communication specifications: a comprehensive study." In: *ACM Computing Surveys (CSUR)* 33.4 (2001), pp. 427–469.

[15] Christian Decker, Jochen Seidel, and Roger Wattenhofer. "Bitcoin meets strong consistency." In: *Proceedings of the 17th International Conference on Distributed Computing and Networking*. 2016, pp. 1–10.

[16] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. "Bitcoin-ng: A scalable blockchain protocol." In: *NSDI*. 2016, pp. 45–59.

[17] Alan Fekete, Nancy Lynch, and Alex Shvartsman. "Specifying and using a partitionable group communication service." In: *ACM Transactions on Computer Systems (TOCS)* 19.2 (2001), pp. 171–216.

[18] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. "Impossibility of Distributed Consensus with One Faulty Process." In: *J. ACM* 32.2 (Apr. 1985), pp. 374–382. issn: 0004-5411. doi: 10.1145/3149.214121. url: https://doi.org/10.1145/3149.214121.

[19] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. "Algorand: Scaling byzantine agreements for cryptocurrencies." In: *OSDI*. 2017, pp. 51–68.

[20] Yue Guo, Rafael Pass, and Elaine Shi. "Synchronous, with a Chance of Partition Tolerance." In: *CRYPTO*. 2019, pp. 499–529.

[21] Kendric Hood, Joseph Oglio, Mikhail Nesterenko, and Gokarna Sharma. "Partitionable Asynchronous Blockchain." In: *32nd ACM Symposium on Parallelism in Algorithms and Architectures* (2020, submitted for publication).

[22] Kolbeinn Karlsson, Weitao Jiang, Stephen Wicker, Danny Adams, Edwin Ma, Robbert van Renesse, and Hakim Weatherspoon. "Vegvisir: A partition-tolerant blockchain for the internet-of-things." In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2018, pp. 1150–1158.

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

[24] S King and S Nadal. "Peercoin–secure & sustainable cryptocoin." In: *Aug-2012 [Online]. Available: https://peercoin. net/whitepaper ()* (2012).

[25] Andrei Kirilenko, Albert S Kyle, Mehrdad Samadi, and Tugkan Tuzun. "The flash crash: High-frequency trading in an electronic market." In: *The Journal of Finance* 72.3 (2017), pp. 967–998.

[26] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem." In: *ACM Trans. Program. Lang. Syst.* 4.3 (July 1982), pp. 382–401. issn: 0164-0925. doi: 10.1145/357172.357176. url: https://doi.org/10.1145/357172.357176.

[27] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine generals problem." In: *Concurrency: the Works of Leslie Lamport*. 2019, pp. 203–226.

[28] Chenxing Li, Peilun Li, Dong Zhou, Wei Xu, Fan Long, and Andrew Yao. "Scaling nakamoto consensus to thousands of transactions per second." In: *arXiv preprint arXiv:1805.03870* (2018).

[29] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. "A secure sharding protocol for open blockchains." In: *CCS*. 2016, pp. 17–30.

[30] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. "The Honey Badger of BFT Protocols." In: *CCS*. Vienna, Austria: ACM, 2016, pp. 31–42. isbn: 978-1-4503-4139-4. doi: 10.1145/2976749.2978399. url: http://doi.acm.org/10.1145/2976749.2978399.

[31] Louise E. Moser, Yair Amir, P. M. Melliar-Smith, and Deborah A. Agarwal. "Extended Virtual Synchrony." In: *ICDCS*. 1994, pp. 56–65.

[32] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system,"* 2008.

[33] Rafael Pass and Elaine Shi. "Fruitchains: A fair blockchain." In: *Proceedings of the ACM Symposium on Principles of Distributed Computing*. 2017, pp. 315–324.

[34] Rafael Pass and Elaine Shi. "Hybrid consensus: Efficient consensus in the permissionless model." In: *DISC*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2017.

[35] Rafael Pass and Elaine Shi. "Thunderella: Blockchains with optimistic instant confirmation." In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2018, pp. 3–33.

[36] Serguei Popov. *The Tangle*. Tech. rep. https://iota.org/IOTA_Whitepaper.pdf. The IOTA Foundation, 2018.

[37] André Schiper. "Dynamic group communication." In: *Distributed Computing* 18.5 (2006), pp. 359–374.

[38] André Schiper and Sam Toueg. "From set membership to group membership: A separation of concerns." In: *IEEE Transactions on Dependable and Secure Computing* 3.1 (2006), pp. 2–12.

[39] Yonatan Sompolinsky and Aviv Zohar. "Secure high-rate transaction processing in bitcoin." In: *International Conference on Financial Cryptography and Data Security*. Springer. 2015, pp. 507–527.

[40] Jason A. Tran, Gowri Sankar Ramachandran, Palash M Shah, Claudiu Danilov, Rodolfo A. Santiago, and Bhaskar Krishnamachari. "SwarmDAG: A Partition Tolerant Distributed Ledger Protocol for Swarm Robotics." In: *Ledger*. 2019.

[41] Harald Vranken. "Sustainability of bitcoin and blockchains." In: *Current opinion in environmental sustainability* 28 (2017), pp. 1–9.

[42] Gavin Wood. "Ethereum: A secure decentralized generalized transaction ledger." In: *Ethereum project yellow paper* 151 (2014), pp. 1–32.

[43] Yang Xiao, Ning Zhang, Wenjing Lou, and Y. Thomas Hou. *A Survey of Distributed Consensus Protocols for Blockchain Networks*. 2019. arXiv: 1904.04098 [cs.CR].

[44] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. "RapidChain: A Fast Blockchain Protocol via Full Sharding." In: *IACR Cryptology ePrint Archive* 2018 (2018), p. 460.