Transaction Finality through Ledger Checkpoints

Ratul Antik Das
Research and Development
Kona Software Lab
Dhaka, Bangladesh
ratul.antik@konasl.com

Md. Muhaimin Shah Pahalovi
Research and Development
Kona Software Lab
Dhaka, Bangladesh
muhaimin.shah@konasl.com

Muhammad Nur Yanhaona
Research and Development
Kona Software Lab
Dhaka, Bangladesh
nur.yanhaona@konasl.com

***Abstract*—The reversal of transactions due to blockchain ledger reorganization has become a major hindrance for public blockchain technologies' adoption in real-world business and financial applications. Since a typical real-world product, service, or agreement cannot be reversed; associated transactions in the blockchain ledger must also be final. This paper describes and analyzes the transaction finality solution for our proof of work (PoW) mining based blockchain network, the Kona Blockchain Platform. The solution works even if 49% of the network's total mining power is compromised due to malicious attacks. Although designed for our specific platform, the ideas from the solution can be easily adapted to achieve transaction finality in existing public blockchain networks. This paper also discusses how this can be done. To the best of our knowledge, ours is the first solution for deterministic transaction finality for blockchain networks incentivized exclusively by PoW mining.**

***Index Terms*—Computer Networks, peer-to-peer computing, distributed information systems, Fault tolerance, Protocols**

I. Introduction

Since the publication of Satoshi Nakamoto's 2008 seminal paper [23] that begot it, the blockchain technology has spurred a flurry of commercial activities. Blockchain-based digital currencies, called *cryptocurrencies* [5] [23], have gained worldwide interest. Blockchain smart contracts [28] [26] with their capability to encode and enforce the rules of interactions among mutually untrusted parties have led to the development of many innovative real-world applications. Some pundits even hail the blockchain technology as a revolution that can transform the global economy [4].

From a technological standpoint, a blockchain system is a decentralized network of peers who maintain the state of a distributed ledger – *the blockchain* – by each executing (or validating) all transactions sent to it. The current state of the ledger is updated by adding a block of transactions. Each block has a pointer to its previous block; hence forming a chain called blockchain. In Nakamoto's proposal [23], a peer who wants to append a valid block in the blockchain has to solve a computational puzzle whose solution is difficult but verification is cheap. The solution of the puzzle becomes a part of the block that the peer is trying to append in the blockchain. This serves as a *Proof of Work* (PoW) [6] of the peer to the rest of the network. Combined with the blockchain data structure, PoW provides a simple basis for consensus [7] among the network peers about the state of the ledger.

This research is funded by KONA Software Lab Ltd.

Users of a peer-to-peer network send transaction requests to arbitrary subset of peers. It is theoretically impossible to ensure that all transactions will reach all peers within a defined time interval given that mere network connectivity among the peers cannot be guaranteed [16]. Consequently, a deterministic ordering of the requested transactions is also unachievable. Given this problem, each peer of the blockchain network arbitrarily picks among the transactions it received from users to process and tries to add blocks in its local version of the chain using those transactions. Whenever it receives a valid blockchain from any of its network neighbors that is longer – i.e., has more work being done on it – than its own version, it accepts the longer chain as canonical and tries to redo left-out transactions in the new canonical chain.

Recent alternatives to the PoW based consensus protocol for public blockchain networks such as *Proof of Stake* (PoS) [2] or *Proof of Elapsed Time* (PoET) [10] work under the same principal. These protocols change the block construction process and the criteria for deciding the best among candidate blockchains. The possibility of a better blockchain replacing a peer's current local chain remains regardless of their protocol differences. Consequently, the long term persistence of blockchain transactions is always a probabilistic guarantee. Currently, guaranteed permanence – commonly called *transaction finality* – in blockchain technology is available only under stringent trust and connectivity constraints among the network peers [11], [21]. These constraints are typically difficult to apply in scalable, peer-to-peer, public networks.

However, the probabilistic permanence of transactions is a serious hindrance for blockchain technology adoption in modeling real-world business and financial interactions. This is because, unlike blockchain transactions, the consequences in the real world are often irreversible or cannot be reproduced. Hence the transaction finality problem must be solved to realize blockchain technology's potential to document the real-world interactions and for cryptocurrencies' acceptance in all kinds of payments.

This paper describes the transaction finality solution of the *Kona blockchain platform*: a blockchain solution that supports smart contracts with additional features needed to carry financial and business applications on a peer-to-peer blockchain network incentivized by PoW mining. Finality is achieved by periodically establishing a consensus across the network on *accepted, irreversible state* of the blockchain ledger we call a *checkpoint*. All transactions up to the latest checkpoint are final, thus, can be used safely for any real-world decision making. Transactions mined in blocks after the latest checkpoint are only probabilistically secure until they are included into the next checkpoint.

The process of checkpoint establishment which we call *the checkpoint protocol* neither compromises the PoW mining protocol nor does it necessitate any new network connectivity constraints among the mining peers. Miners' collaboration during checkpoint establishment is incentivized by a PoW mining based voting scheme. The only addition to the network is a *byzantine fault-tolerant* (BFT) [20] support service that disburses checkpoint ballots, seals checkpoint blocks, and does nothing else. The presence of the support service also ensures that a large group of new miners entering the network after a checkpoint cannot reverse a check-pointed state retroactively. We prove that under the current PoW incentive scheme, the addition of such service is mandatory to ensure transaction finality in a peer-to-peer network. It can be proven that our checkpoint protocol is fair, non-exclusionary, and can only be interrupted by a successful DDoS attack [29] on the support service.

The paper provides a discussion of the design motives and a detailed analysis of the qualitative properties along with the description of the checkpoint protocol. In addition, the paper illustrates how a byzantine fault-tolerant support service can be realized and properly incentivised so that it cannot willfully bias or destabilize the checkpoint protocol. The rest of the paper is organized as follows:

A. Paper Organization

Section II describes related work on transaction finality in blockchain technology; Section III analyses a PoW incentive scheme's impact on miner collaboration and consequent challenges in achieving transaction finality; Section IV then establishes the model for the check-pointing problem and elaborates on modeling objectives; Section V presents our checkpoint protocol; Section VI analyses how the protocol meets the objectives of Section IV; Section VII then touches on some implementation concerns, in particular, it describes how to implement a BFT support service and discusses what can be its alternative in existing public blockchain networks; Finally, Section VIII concludes the paper along with a discussion on future improvements to the protocol.

II. Related Work

It has been theoretically proven that purely peer-to-peer blockchains are probabilistic state machines that cannot provide any finality guarantee [25]. The proof follows easily from the fact that malicious peers can reach 51% majority in a network and reverse all transactions. So any discussion of blockchain ledger stability assumes that the majority mining power is held by the honest peers. A more practical analysis [24] shows that assuming a bounded network delay and no message loss, a blockchain network achieves consistency (i.e., transaction finality) with probability approaching 1. The problem is messages can be lost and the network delay bound cannot be predetermined in practical peer-to-peer network even with majority honest peers. Hence, *probabilistic* transaction finality is the best that can be achieved without any new restriction in the blockchain network.

However, existing blockchain consensus protocols that claim to provide transaction finality impose restrictions on network participation that are difficult to realize in a peer-to-peer setting where participants can join and leave at any time. For example, the Ripple [11] protocol assumes that 80% of the neighbors of each honest peer are honest and malicious neighbors cannot intercept communications between honest peers. Similarly, the Stellar consensus protocol [21] assumes that the majority honest peers of the network form a connected subnet that provides guaranteed message delivery. The Tendermint consensus protocol [8] also assumes $\frac{2}{3}$ majority of honest peers receive each message within a bounded time delay. Under this assumption, it provides a *PBFT* [9] solution decentralized for blockchain technology domain. The central issue with all these protocols is that they assume a gossip communication protocol [3] can be utilized as a means to implement a reliable broadcast primitive [18] in a peer-to-peer network. This assumption is not theoretically valid. Hence these protocols suffer stagnation when broadcast fails.

The forerunner of the *business blockchain movement*, Hyperledger, provides transaction finality for two of its blockchain solutions: *Iroha* [22] and *Fabric* [17]. Both assume there is a BFT ordering service that orders and reliably delivers messages to all network peers. The network is definitely not peer-to-peer in either case and so far no convincing BFT solution for a distributed ordering service has been proposed.

None of the aforementioned protocols work on a network incentivised by PoW mining. The only notable transaction finality solution for PoW blockchain networks is Ethereum's Casper [2]. Casper assigns validator nodes (based on deposit of stakes) among the mining population to regularly vote on alternative blockchain versions. The version receiving $\frac{2}{3}$ majority of votes is called a finalized checkpoint, thus irreversible. Since the underlying networking characteristics remain the same, $\frac{2}{3}$ validator nodes may fail to collaborate to establish a voting consensus. Thus, in the worst case, no checkpoint can be finalized.

A major difference between ours and existing transaction finality solutions is that we do not assume the existence of any reliable broadcast primitive in the underlying peer-to-peer network. Still, we periodically establish checkpoints in the blockchain ledger. In addition, although we employ voting for reaching consensus about a checkpoint like others, our voting process is also governed by a PoW incentive unlike any.

III. An Analysis of PoW Incentive

From a technical perspective, Nakamoto's Bitcoin [23] took the world by storm because it shows for the first time that a highly secure distributed system can be built from laissez-faire collaboration of a network of mutually-distrusted autonomous peers. Even continuous network connectivity among the peers is not required and the network-wide connection topology remains dynamic. He deemed a PoW for block mining is necessary to thwart the traditional networking attacks such as the Sybil Attack [14] on such a system. To successfully alter a shared blockchain ledger of honest miners, an attacker has to surpass their combined mining power to construct an alternative chain that is longer; then offer it to them for synchronization. The effort becomes more futile as the population of honest peers increases.

The strength of information security in a blockchain network is proportional to the mining capacity of its honest peers. Therefore, since its inception, keeping honest peers interested in network participation is a central concern in blockchain technology. Nakamoto's ingenious idea was to make participation in the block mining process an economic activity [19] by rewarding a block miner for his/her PoW in terms of transaction fees and a block reward. This is the crux of the PoW incentive scheme.

PoW incentive makes honest behavior the rational behavior [23] in a blockchain network governed by economic principles. This particular feat is the source of its resilience. So far no alternative to PoW incentive is proven to be equally scalable, secure, and censorship-resistant. Consequently, the largest and most-powerful blockchain networks remain PoW networks despite widespread scrutiny of PoW mining's power consumption cost [1]. Hence, transaction finality solutions should also target PoW mining based blockchain networks.

In the PoW incentive scheme, all miner activities including computation and communication are governed by the rational behavior of maximizing economic gain. As a result, in such a network a consensus on an irreversible ledger state (i.e., transaction finality) cannot be established without halting the mining process, as explained in the following lemma:

Lemma III.1. *Transaction finality is unachievable without halting block mining in a PoW blockchain network.*

Proof. Since there is no benefit for a peer in throwing away its local blockchain ledger version and accepting a neighbor's blockchain that has the same amount of PoW being done on it, information on equally good alternative blockchain versions will not be spread across the network. Consequently, at any instance of time there might be as many equally good blockchain ledger versions as the number of miners. In addition, a peer cannot accurately estimate the number of peers currently active in the network. Consequently, it cannot determine if its local version of the blockchain ledger is accepted by the majority.

Since a rational miner can convincingly neither deduce if its own attempt to establish a network-wide consensus about irrefutable blockchain ledger state will be successful nor deduce whether it will be worthwhile to throw away its own equally good local ledger state in response to a request from a neighbor, the only rational behavior is to keep lengthening its own local ledger version. In the worst case, all miners will emulate the same behavior and the only consented state will be the genesis state and no transaction can ever be declared final. Hence halting block mining is essential to force a ledger state consensus. □

In what state of the blockchain ledger the next checkpoint consensus should be attempted must be known to all network peers beforehand; otherwise they will not know when to halt in the absence of a reliable broadcast mechanism. A simple way to achieve this is to make checkpoints periodical to the length of the blockchain or the total amount of PoW. Another alternative is to decide the next checkpoint time as part of establishing consensus for the current checkpoint.

An important property of the transaction finality in a PoW mining based blockchain network is that given it is a requirement, the rational behavior is to halt and establish a checkpoint at intended time before attempting further progress in advancing the blockchain. This is explained in the following lemma (proof omitted):

Lemma III.2. *The only rational behavior is to collaborate on a checkpoint consensus when a miner's local blockchain ledger approaches the checkpoint state.* □

Lemma III.1 and III.2 suggest that a reasonable strategy to achieve transaction finality in a PoW mining based blockchain network is to periodically alternate between a block mining and a checkpoint protocol. However, the checkpoint protocol should be designed with care to avoid scalability and fairness issues during its execution and to avoid introducing security, and censorship issues in the block mining process during the protocol switching.

IV. Problem Modeling

From the discussion of Section III, we understand that the goal of the checkpoint protocol is to achieve a network-wide consensus on an irreversible and unique state of the blockchain ledger at deterministic intervals. We also understand that at the end of a checkpoint interval, there might be many miners holding different blockchain ledgers that are valid candidates to be the next checkpoint and many who are lagging behind. If we call the formers, *Front-runners*, the qualitative objective of the checkpoint protocol should be as follows:

> **Fairly select** a single ledger version from the front-runners **without censoring** the lagging behind miners and ensure **unaffected progression** of PoW block mining after selection.

Our approach to the aforementioned highlighted objectives is to establish each checkpoint based on a majority vote by the currently active network peers on the chains of the front-runners who reached the checkpoint candidacy state the earliest. The evidence of checkpoint selection is included in the winning chain as a checkpoint block and the proceed for mining the checkpoint block is distributed to the peers voted to support it. Here a vote is casted for a candidate by solving PoW puzzles on its chain state.

Assume there are total N active peers in the blockchain network and the current state of the local blockchain ledger version of $Peer\ i$ is represented by $BS_i(h, l, t, c)$ where h is the current header block hash, l is the length of the blockchain, t is the header block mining time, and c is the last checkpoint block hash. Further, let $BS_i^h, BS_i^l, BS_i^t, BS_i^c$ denote the individual attributes of $Peer\ i$'s state and $BS_i^h(n)$, $BS_i^t(n)$ refer to the header block hash and mining time when its ledger length was n. In addition, let $\Lambda(h)$ returns the length of the blockchain ledger and $\mathcal{T}(h)$ the time when the block with hash h was mined by anyone. Finally, let C denotes the set of checkpoints. Then the objective of our checkpoint protocol is to maintain the following invariants as true for all currently active network peers:

$$BS_i^c = BS_j^c,\ \forall i \neq j\ \&\ i.j \in N \tag{1}$$

$$\mathcal{T}(c) = \min_{i \in N}\{(\mathcal{T}(BS_i^h(l)))\ |\ l = \Lambda(c)\},\ \forall c \in C \tag{2}$$

$$\frac{\sum_{i=1}^{N}\{1\ |\ BS_i^h(l) = c,\ l = \Lambda(c)\}}{N} \geq .51,\ \forall c \in C \tag{3}$$

Invariant 1 says that all active network peers advance their ledger versions from a common check-pointed state, *Invariant 2* ensures that each checkpoint is selected among the ledger versions that reached the checkpoint candidacy state the earliest, finally, *Invariant 3* dictates that the candidate ledger version that gains 51% majority support (i.e., synchronized by the majority) becomes the checkpoint.

The core underlying concerns related to maintaining these invariants are measuring the current status of the network (for estimating N and comparing BS_i^t values), ensuring information propagation in the network for voting based consensus establishment and sealing of a checkpoint block for the permanence of the consented ledger state. The following subsections address these issues and associated matters.

A. Network Population Estimation

In a purely peer-to-peer blockchain network, no peer has an accurate estimate of the size of the currently active peer population, that is, the value of N. Thus we introduce a set of *support service* nodes for active network population estimation. The addresses of these nodes are known to the mining peers. The support service nodes, or support nodes, form a distributed population status estimation service. Each mining peer exchanges periodic heartbeat messages with random support nodes. To be considered currently active and eligible for participation in the upcoming checkpoint consensus protocol, a peer must have exchanged a heartbeat with some support node within a defined time window we call the *keep-alive time interval*. Assume the time-stamp of the latest heartbeat message of $Peer\ i$ is H_i^t, the keep-alive time interval is Δ, and the network time of the distributed support service is Υ. Then the rule for estimating N is as follows:

$$N = \sum_{i=0}^{i=\infty} 1\ |\ \Upsilon - H_i^t \leq \Delta \tag{4}$$

Note that all the attributes of $Peer\ i$'s ledger state are self-evident, except BS_i^t: the mining time of the header block. The peers of a blockchain network are only very loosely time-synchronized [27] and a mining peer can easily advance its clock to gain advantage in the checkpoint selection process. To tackle arbitrary adjustments of the block mining time, BS_i^t is derived from $Peer\ i$'s heartbeat message timing. Peers' heartbeat messages bear their header block hash and support nodes' acknowledgements for those heartbeats bear an acknowledgement time-stamp. If the heartbeat message sequence of $Peer\ i$ is $1, 2, \cdots M$, and $\beta(i, j)$ returns the block hash of j^{th} message and $\Gamma(i, j)$ the acknowledgement time-stamp of that message then:

$$BS_i^t = \min_{j \in [1,M]}\{\Gamma(i, j) | \beta(i, j) = BS_i^h\} \tag{5}$$

The formulation of *Equations 4* and *5* makes exchanging periodic heartbeat messages with support nodes a rational behavior for the mining peers.

B. Checkpoint Block Sealing

The support service also seals the checkpoint block by signing it once a majority consensus on the winning ledger version is reached. This seal is needed to ensure that even if the entire population of active mining peers is replaced, new peers cannot reverse a check-pointed state. A sealed checkpoint block is a 7-tuple of the form $\langle \zeta_h, \zeta_c, \zeta_t, \zeta_e, \zeta_v, \zeta_i \rangle$ where ζ_h is the block hash of the check-pointed ledger state, ζ_c is the current checkpoint interval counter, ζ_t is the time-stamp of the checkpoint block, ζ_e is the evidence that the estimation of N is accurate, ζ_v is the evidence of the majority's support for the checkpoint, and ζ_i is the next checkpoint interval length. ζ_c and ζ_t ensure that the support service cannot regress to an earlier state of the blockchain and resume checkpoint sealing from there, and ζ_i makes provision for dynamic adjustments of the checkpoint interval.

Introduction of the support service raises the concern that support service nodes may skew the voting process for checkpoint consensus and, consequently, compromise *Invariant 1* and *2*. As a check against such manipulation, we limit what support service knows about the consensus process and adopt the following principle:

> The support service should know about a checkpoint consensus process only after its inception and it must not know how the peers voted until the termination of a voting cycle.

The idea is that the front-runner peers should initiate the checkpoint consensus process. If it starts due to some support service action then the service can give preference to some specific front-runner by manipulating the heartbeat acknowledgement time, consequently affecting BS_i^t. [1]

The termination constraint is required so that the support service is not tempted to drop evidences of vote for a specific chain and refuses to seal the checkpoint block if its desired front-runner is not the winner. The termination of a voting cycle must be detected and incorruptible evidence of voting

[1]The support service cannot determine BS_i^l from the change of BS_i^h in heartbeat messages because any number of blocks may be added in *Peer i*'s ledger between two successive heartbeat messages.

decisions must be registered and shared before the support service can interpret the outcome.

C. Motivating Information Propagation

Since all peer actions are governed by economic motives in a PoW mining based blockchain network, encouraging peer collaboration during the checkpoint establishment is an important concern. The checkpoint invariants mentioned before are insufficient in that regard because they only dictate the requirements – not how to collaborate in achieving them.

In particular, when front-runner mining nodes reach the next checkpoint target state in their respective blockchain ledgers, they are motivated to initiate a checkpoint election process. Their lagging behind neighbors are motivated to participate in the process for their own survival. However, there is no incentive for the lagging behind peers to further spread the news of the ongoing election. Consequently, a front-runner peer initiated checkpoint election process may never be heard by the majority peers, let alone reach a consensus.

We tackle this problem by incentivising information propagation specifically during the checkpoint election. Lagging behind neighbors get the rights to vote on front-runners' blockchains not only because they are currently active but also because they have received tokens (or ballots) from the latter. A voting peer then creates sub-tokens from its token and propagates them to its neighbors. A hierarchy of sub-tokens can be created in this manner based on the idea of hierarchical credential delegation presented in [12]. A voting peer registers its vote by submitting the token (or sub-token) of its choice to the support service.

If a token and its sub-tokens are increasingly labeled according to the depth of the delegation path and the total reward for casting a vote on the checkpoint winner blockchain ledger is F then the reward for casting a specific sub-token of depth k in favor of the winner blockchain is distributed according to the following formula:

$$f^k_{[i]} = \begin{cases} F \times C(1-C)^{i-1} & \forall i < k \quad (6a) \\ F \times (1-C)^{k-1} & i = k \quad (6b) \end{cases}$$

Here $f^k_{[i]}$ represents the reward for the i^{th} peer on the token delegation path and C is any suitably chosen fractional constant. Sybil attack resistance [14] and game theoretic soundness of this reward distribution scheme is discussed by the scheme's authors in [15].

V. Checkpoint Algorithm

The general description of the checkpoint protocol is as follows:

1) If a miner, f_i, reaches the checkpoint candidacy state, it creates a voting token $t_{f_i}(0)$ (0 representing the token delegation depth) from its latest heartbeat acknowledgement that proves its $BS^t_{f_i}$.
2) For each neighbor peer p, f_i creates a time-stamped sub-token $t^p_{f_i}(1)$ dedicated for p and requests a vote. This initiates a checkpoint consensus voting round.
3) A network peer p evaluates all voting requests, $t^p_{f_i}(d_{f_i})$s, it has received, validates the blockchain ledgers of the corresponding candidates, determines voting for which candidate maximizes its own profit, then casts an encoded vote for that candidate, f_c, by sending a payload with its next heartbeat message to the support service. Peer p receives a vote acceptance acknowledgement VA^t_p in return.
4) A front-runner miner f_i makes its own encoded vote from $t_{f_i}(0)$ and registers the vote after it receives acceptance notifications from some neighbors or after a maximum waiting time.
5) Any peer p who has voted keeps reaching out for more lagging behind neighbors by sending them voting sub-tokens made of its own $t^p_{f_c}(d_{f_c})$ and VA^t_p.
6) Until the end of the voting round, peers can keep changing their votes as they hear of better alternative to their current choice.
7) At the end of the voting round, the peers reveal their vote to support service by supplying the decoding key for their encoded vote with their next heartbeats.
8) If there is a single majority, the support service supplies sealing materials for the checkpoint block that the wining majority mines. The remaining others synchronize their ledgers with the majority and everyone switches back to the block mining phase.
9) If there is no single majority then some inferior candidates are filtered using a universally known, deterministic, and fair criterion. A new voting round begins with fewer candidates. The cycle continues in this manner until a single majority consensus is reached.

Listing 1 presents a redacted pseudo-code of the network peers' algorithm for the checkpoint consensus protocol. The pseudo-code does not show any error processing or malicious behavior detection.

```
func votingRound(currC, currVCert, round) {
    knownCandidates = {}
    // if peer's selected candidate is not eliminated in the last round
    // then starts the new round with the selected candidate
    if (currC != nil) knownCandidates = {currC.candidate}
    // based on a heartbeat message acknowledgement counter peer should
    // know when the voting round should be ended
    while(roundNotEnded()) {
        // get the vote requests received since the last heartbeat
        S = getNewCandidates()
        if (empty(S)) {
            // if no request received then check if the support
            // service has provided some candidate info
            stat = getLastHeartbeatStat()
            if (stat.candadate != nil) {
                S = {probeCandidate(stat.candidate, stat.probingToken)}
            }
        }
        // in case there are malicious front-runners that are
        // eliminated in the last round but still seeking votes, do
        // a sanity filtering of incoming candidate set
        F = filterCandidateByRound(S, round)
        b = selectBest(F)
        if (b != nil && !contains(knownCandidates, b)
                && (currC = nil || isBetter(b, currC))) {
            // change vote if the new candiate is better
            updateLedger(b)
            currC = b
            currVCert = castVote(b)
30             knownCandidates =  knownCandidates + {b}
31             // encourage all neighbors to switch to the chosen
32             // candidate by sending them sub–token
33             seekVotesFromNeighbors(b, getAllNeighbors())
34         } else if (currC != nil) {
35             // keep the current vote intact with the support service
36             currVCert = retainVote()
37             // if there is any new peer connections then influence
38             // them to support the chosen candidate
39             nn = getNewNeighbors()
40             seekVotesFromNeighbors(currC, nn)
41         }
42     }
43     // reveal the encoded vote to the support service at the end of round
44     revealVote(currC)
45     return currVCert
46 }
47
48 func waitForConsensus(initialVoteCert, initialToken) {
49     round = 0
50     // a front–runner  will start with its own ledger version and token
51     currToken = initialToken
52     currVCert = initialVoteCert
53     while (true) {
54         // complete a voting round
55         lastVoteCert = votingRound(currToken, currVCert, round)
56         // check for majority consensus
57         winner = verifySingleMajority()
58         if (winner != nil) {
59             // if consensus is reached then sync the chain, get the
60             // checkpoint block, and break the loop
61             if (latVoteCert.candidate == winner) {
62                 block = getCheckpointBlock()
63                 addCheckpointBlock(ledger, block)
64                 break
65             } else {
66                 syncWithWinner(winner)
67                 break
68             }
69         } else {
70             // otherwise prepare for the next round
71             round++
72             if (eliminated(lastVoteCert, round)) {
73                 currToken, currVCert = nil
74             }
75         }
76     }
77 }
78
79 func roundNotEnded() {
80     // retrieve information from the latest heartbeat acknowledgement
81     stat = getLastHeartbeatAckStat()
82     // if support service has not declared that checkpoint protocol has
83     // started then checkpoint voting can continue for arbitrary long
84     if (stat.mode != CHEK_POINTING_INITATED) return true
85     // otherwise round continues until the support service checkpoint
86     // round counter reaches 0
87     return (stat.counter > 0)
88 }
```

Listing 1. A miner's perspective of the checkpoint protocol

Note that a voting round is self-terminating. Each miner individually determines when to stop voting and reveal its final candidate of choice based on a support service clock counter (*Line 79*). In addition, the whole consensus process is guaranteed to converge as each voting round reduces the front-runner candidates count and ensures that all honest network peers get to know about all the remaining candidates.

Listing 2 presents a redacted pseudo-code of the support service side of the checkpoint algorithm. The vote revelation and error processing related logic are omitted again.

```
func miningModeHeartbeatProcessor(heartbeat, miner) {
    // if the miner is earlier detected to be malicious ignore it
    if (blacklisted(miner)) return NACK
    if (powVerificationFailed(miner, heartbeat)) { // verify PoW
        blacklistMiner(miner)
        return NACK
    }
    // a peer can send encoded vote when others are in block mining mode
    if (heartbeat.type == CHECKPOINT_VOTE) {
        // to record a vote the peer must be considered active beforehand
        if (!recordedAsActive(miner)) {
            return NACK
        }
        // update the last communication time of the peer
        updateLastExchangeTime(miner)
        // record the vote and generate an acknowledgement certificate
        vCert = recordEncodedVote(heartbeat.vote)
        // if majory recorded votes, go to checkpoint consensus process
        N = getMinerCountEstimate()
        c = countCastBallots()
        if (c >= N/2) {
            initiateCheckpointConsensus()
        }
        // generate challenge text for vote change PoW
        chal = generateVoteChangeChallenge(vCert, miner)
        // send acknowledgement
        return new VoteAccecptAck(vCert, chal)

    } else  {
        // update the last communication time
        updateLastExchangeTime(miner)
        // send acknowledgement for heartbeat message
        ack = generateAck(heartbeat)
        chal = generateHeartbeatChallenge(heartbeat, miner)
        return new HeartbeatAck(ack, chal)
    }
}


func initiateCheckpointConsensus() {
    // Once checkpoint protocol has initiated; no new front–runner can be
    // a candidate for checkpoint after a certain time
    scheduleCandidateListFreeze(ACTIVATION_WINDOW)
    // launch an alternative heartbeat message processor when in the
    // checkpoint consensus mode
    setHeartbeatProcessor(checkpointHeartbeatProcessor)
    // first voting round will continue for N− 1 ticks of clock timer
    N = getMinerCountEstimate()
    votingRoundTerminator = initCountDownCounter(N− 1)
    consensusEstablished = false

    do {
        // wait for the voting round to end
        waitForReachingZero(votingRoundTerminator)
        // set the heartbeat processor to vote decoding mode
        setHeartbeatProcessor(voteRevealHeartbeatProcessor)
        // wait for vote revelation to end
        voteRevelationCounter = initCountDownCounter(ACTIVATION_WINDOW)
        waitForReachingZero(voteRevelationCounter)

        stat = checkForSingleMajority()
        if (stat == true) {
            // in case a single majority is detected then consensus is
            // established; publish checkpoint block sealing material
            // to be retrieved by the mining nodes
            publishResultWithSealingMaterial()
            consensusEstablished = true
        } else {
            // if no single majority consensus is reached in this round
            // then publish the verifiable filtering criteria
            publishResultWithCandidateFilter()
            // reset the vote collection counter to allow N− 1 ticks
            // for the next round
            resetCounterTo(N− 1)
            // reset the heartbeat processor to vote recording mode
            setHeartbeatProcessor(checkpointHeartbeatProcessor)
        }
    } while(!consensusEstablished)

    // reset database and advance to the checkpoint interval
    clearVoteDatabase()
    updateCheckpointCounter()
    // resume normal interaction
    setHeartbeatProcessor(miningModeHeartbeatProcessor)
}

func checkpointHeartbeatProcessor(heartbeat, miner) {
    // blacklisted peers and peers recorded as inactive before cannot
89      // participate in the checkpoint establishment process
90      if (blacklisted(miner) || !recordedAsActive(miner)) return NACK
91      // verify PoW
92      if (powVerificationFailed(miner, heartbeat)) {
93          blacklistMiner(miner)
94          return NACK
95      }
96      // update the last communication time of the peer
97      updateLastExchangeTime(miner)
98      // get the next existing voter from miner specific permutation
99      // of the voting population to suggest as a candidate
100     suggestion = getNextFromExistingVotersOrder(miner)
101     // heartbeat from a lagging behind mining peer who does not know
102     // about ongoing check-pointing process
103     if (heartbeat.type = MINING_HEARTBEAT) {
104         // update the last communication time of the peer
105         updateLastExchangeTime(miner)
106         // get information about voting round terminator counter
107         stat = getCounterStatistics()
108         // send reply
109         ack = generateAck(heartbeat)
110         chal = generateHeartbeatChallenge(heartbeat, miner)
111         return new HeartbeatAck(ack, chal, stat, suggestion)
112     } else {
113         vCert = recordEncodedVote(heartbeat.vote)
114         chal = generateVoteChangeChallenge(vCert, miner)
115         stat = getCounterStatistics()
116         return new VoteAccecptAck(vCert, chal, stat, suggestion)
117     }
118 }
```

Listing 2. Support service's perspective of the checkpoint protocol

Any heartbeat exchange with the support service involves solving a new PoW puzzle. This is done to avoid a denial of service attack on the support service by frequent heartbeats [6]. In addition, changing an existing vote involves solving increasingly more difficult PoW challenge (*Line 25* and *114*). This strategy compels rational miners to be prudent with their vote change decisions.

The support service alternates between different heartbeat processors (*Line 46, 56, 76*, and *84*) based on an internal clock and the state of the ongoing checkpoint consensus process. In particular, each voting round remains open for $N-1$ clock ticks (*Line 49* and *74*). Successive ticks of the internal clock should provide enough time for a blockchain ledger synchronization between a pair of interacting network peers and the *activation window*, Ω, of *Line 58* should be large enough to allow all network peers to exchange at least one heartbeat message with some support service node.

The support service freezes the checkpoint candidate list Ω time after locally initiating the checkpoint protocol (*Line 43*). Observant readers will notice that it does so without even knowing the candidates, as all votes are encoded. This operation finalizes the list of peers to be probed by lagging behind miners who did not vote yet.

VI. Fitness Analysis of Checkpoint Protocol

In this section, we discuss several properties of the checkpoint protocol. Throughout this discussion we assume that 51% of the network peers are honest.

Since the mining peers determine the end of a voting round based on a support service clock counter, it is guaranteed that each voting round will terminate at a deterministic time. What remains to be proven is that the support service can unequivocally determine that some front runner peers have reached the checkpoint candidacy state, its strategy to freeze the front-runner checkpoint candidate set (*Line 47*) is not exclusionary, and its round termination counter provides enough time for all rational miners to make a final voting decision in each round. Following lemmas address these concerns:

Lemma VI.1. *The support service can determine the initiation of a checkpoint consensus voting process without knowing the identities of honest miners.*

Proof. The particular problem with checkpoint protocol initiation detection is that the front-runner miners launch the consensus voting process by directly requesting their neighbors to vote without informing the support service anything about their ledgers' checkpoint candidacy state. Keeping the support service oblivious of the front-runners is important to avoid introducing any support service induced bias on Invariant 2 of the checkpoint protocol through BS_i^t values. This in turn creates the problem that malicious peers can pretend that someone has reached checkpoint candidacy state by registering encoded checkpoint vote with support service even when there is no valid checkpoint candidate.

However, since an honest miner will always verify the ledger of a peer requesting checkpoint vote before making a voting decision, a colluding party of lagging-behind malicious miners cannot convince an honest miner to ever vote in their favor. Since the honest miners are the majority, an invalid checkpoint candidate can never reach consensus. So eventually, some honest miners will become front-runner and one or more honest miners will register their checkpoint vote with the support service. If 50% of the currently active miners casted ballot for checkpoint then there must be at least one honest miner who is either a front-runner or who contains a ledger synchronized from a valid front-runner checkpoint candidate. Therefore, the support service can declare initiation of checkpoint consensus voting without a doubt after receiving that many votes. □

Lemma VI.2. *No new checkpoint candidate with non-zero chance to win consensus for its ledger can appear after Ω time of support service's protocol initiation declaration.*

Proof. By the time the support service declares the initiation of the checkpoint protocol, at least one honest miner has registered vote in favor of some valid checkpoint candidate. Consider the worst case of exactly one honest vote. Suppose that vote is casted in favor of front-runner miner f. Then assume by contradiction that the first vote for another valid candidate f' can appear after Ω time of the protocol initiation declaration.

Note that Ω is large enough for all miners to exchange one heartbeat with the support service. Hence any vote casted for a previously unseen alternative front-runner candidate f' after Ω time of the protocol initiation declaration must satisfy $BS_f^t \leq BS_{f'}^t$. According to *Invariant 2* of check-pointing, such an f' cannot win consensus. A contradiction. □

Lemma VI.3. *$N-1$ support service timer ticks are enough for any rational miner to make a final choice about front-runner checkpoint candidates in all consensus voting rounds.*

Proof. The profit sharing scheme (Sub-section IV-C) for vote casted in favor of the winning candidate makes synchronizing local ledger and supplying voting sub-token to any probing peer a rational behavior. So any honest miner who has voted will respond to a probing request coming from another miner unless it is overloaded. Since the support service uses different miner specific permutations for suggesting candidate front-runners (*Line 105* of Listing 2) to network peers, Each miner should receive either one or no probing request per support service timer tick.

Given the number of front-runner checkpoint candidate C satisfies $C \leq N$, a peer has maximum $N-1$ candidates to evaluate other than its current choice. Given each clock interval of the support service provides enough time for a full chain synchronization, a peer can probe all front-runner candidates and decide its final vote by $N-1$ clock ticks. □

That all honest network peers cast votes in each checkpoint voting round does not guarantee an eventual majority consensus. For example, consider the terminal case that each of the N mining peers is a front-runner checkpoint candidate. Then it is not rational for a miner to vote for anyone but itself. Consequently, the checkpoint protocol will continue to run indefinitely and never converge to a consensus. Hence, we need a criterion to reduce the checkpoint candidates count in each round. The criterion we adopt is as follows:

> **Candidate Filtering Criteria:** In each voting round, the candidate with the worst header block mining time (i.e., BS_i^t) and the least votes will be eliminated. In case there are multiple worst candidates, the candidate with the largest header block hash (i.e., BS_i^h) will be eliminated.

Since BS_i^h values arise at random based on the mined block contents of different candidates and their BS_i^t values are directly comparable, the fairness of the candidate filtering criteria is self-evident. The following lemma proves that the criteria ensures a consensus in a finite time.

Lemma VI.4. *A consensus among the honest network peers is guaranteed within N voting rounds when checkpoint candidates are being filtered with the Candidate Filtering Criteria.*

Proof. We consider the worst case that 49% of the peers are malicious or irrational. If we show that a consensus can be reached even for the worst case attack scenario then it can be reached in all other cases as well.

Since all honest peers vote in each round, there must be at least two candidate ledgers held by honest peers with non-zero votes in the first voting round. Otherwise, majority of the honest miners are already in agreement on a single ledger version and a consensus has been reached.

Assume that the set H of honest checkpoint candidates at the end of the first round is $f_1^h, f_2^h, \cdots, f_p^h$. Since malicious peers can form a single or arbitrary many colluding parties, assume that malicious checkpoint candidates set M is $f_1^m, f_2^m, \cdots, f_q^m$. Any $f_m \in M$ may have actually reached the checkpoint candidacy state or simply pretending. In both cases, its ledger will never be synchronized by any honest peer. In the former case, its ledger will eventually become known to all honest miners and the peer will loose its maliciousness. In the latter case, participation in any synchronization attempt will disclose its maliciousness. The malicious peer f_m can withstand successive elimination rounds by arbitrarily simulating good $BS_{f_m}^t$ value or forming an increasingly large pack among malicious peers. Its presence or elimination does not affect the voting decision of the honest peers.

Since no new candidate can appear after the first voting round and each round eliminates one checkpoint candidate, the protocol cannot run for more than N rounds. We consider only those rounds where some member of H is eliminated. If Round r eliminates candidate $f_i^h \in H$ then all peers that voted for f_i^h must vote for someone in $H - f_i^h$ in Round $r+1$. Thus eliminating an honest candidate only increases the percentage of votes for the remaining honest candidates. So by the end of N^{th} round there must be only one candidate in H with all votes of the honest peers. Therefore, a consensus is established. □

On a side note on consensus, according to the characterization of Dolev et. al. on 'the minimum synchronism needed for distributed consensus' [13], a distributed system with point-to-point communication can reach a consensus state only if the messages are ordered and the processors are synchronous. Observant readers should realize that the checkpoint protocol ensures both message ordering and peer synchronism. In particular, the PoW submission involving any heartbeat message exchange eliminates out-of-order messaging from a peer. The message time-stamps further facilitate a partial ordering of all peers' local ledger states. On the other hand, the requirement to interact with the support service within a defined time window (*the keep-alive time interval*, Δ) to participate in checkpoint consensus makes the peers loosely synchronized in time. Note that the synchronism and the message ordering is guaranteed during the execution of the checkpoint protocol only. The network behaves as an asynchronous distributed system at all other times.

Lemma VI.1 to VI.4 prove that our checkpoint protocol can successfully establish a periodic network-wide consensus on irrefutable ledger states while maintaining the Invariants of Section IV among the active peer population. Concerns remain, however, about the involvement of the support service. In particular, how can the support service obstruct or compromise the checkpoint protocol? The following lemma defines the limits of support service induced attacks on the protocol.

Lemma VI.5. *For authenticated communication, only a denial of service (DOS) attack on the support service can obstruct establishment of a checkpoint consensus through a fair election process.*

Proof. We accept a broader definition of the DOS attacks for the proof. We consider a DOS attack on the support service can be both external and internal. That is the support service can itself refuse to serve requests from mining peers for some internal reasons or it can be attacked by one or more malicious entities in the network so that some mining peers cannot communicate with it.

Note that the support service can never make honest peers to vote for a blockchain ledger version that has not reached the checkpoint candidacy state. This is because the peers individually validate candidate blockchain ledgers before making their voting decision. This limits support service's maliciousness to bias the votes in favor of a particular candidate of choice only. There could be two ways to do this. First, aiding the chosen candidate, c, to achieve an artificially better BS_c^t value than others. Second, falsify the population size parameter N to declare c the checkpoint winner even if it did not really get 51% majority vote from the active peer population.

Since the support service cannot determine if honest miners are approaching the next checkpoint candidacy state, making the BS_i^t values of honest miners worse than BS_c^t by denying heartbeat acknowledgement to honest miners is not a realistic approach.[2] The only alternative is to consistently set past times in the acknowledgements for Miner c's heartbeats. Unfortunately, as the BS_c^t values will be reflected in the blocks c mines, it cannot synchronize its intermediate ledger states with other honest miners without revealing the time anomaly. Hence, Miner c must remain isolated from others, mine all blocks since the last established checkpoint upto the upcoming checkpoint candidacy state, and still reach the candidacy state before the candidate list freezes in the first checkpoint voting round. This is infeasible unless Miner c individually holds 51% mining power of the entire network. Then it would need to bias from the support service.

Falsifying N is impossible as active peers retain heartbeat acknowledgement from support service and they can readily proof that support service is malicious if their status is not included in checkpoint block sealing material ζ_e. Since each vote is encoded, the support service cannot deduce if a vote is casted in favor of Miner c or not either. Albeit the support service can deny acknowledgement for any vote not casted in favor of Miner c during the vote revelation steps to make Miner c the checkpoint winner, it has to do so by declaring active miners inactive. That will compromise its own legitimacy to the majority honest miners.

Since the support service can neither bias the voting process nor alter or drop peers' voting decisions, the only way it can be attacked or itself can affect the checkpoint protocol is by halting checkpoint consensus through refusing services. $\square$

VII. Implementation Concerns

We discussed the theoretical soundness of our checkpoint protocol in the previous section. Now we discuss some important implementation concerns. In particular, two implementation issues merit especial attention: designing a BFT fault-tolerant support service and identifying valid participants for checkpoint consensus voting.

A. Distributed Support Service Implementation

Given its essential role in the checkpoint protocol, the support service cannot be a centralized entity. That would make our entire transaction finality guarantee reliant on a centralized trust, consequently, unacceptable. On the other hand, when designing a distributed support service, we must be careful about not introducing the same transaction finality problem that the support service came to solve. If we ensure that support service nodes are directly connected then a traditional PBFT [9] solution would do. A more interesting solution, however, is to maintain a *Proof of Authority (PoA)* blockchain among the support service nodes for the heartbeat database management.

The support service nodes will be loosely time-synchronized and all aware of a predefined block mining interval for registering heartbeat messages from mining peers into new blocks. A mining peer can reach a random support service node during a particular heartbeat exchange. The peer will send a signed solution of a heartbeat-exchange puzzle constructed from the last block of heartbeat registry chain the peer is aware of and the peer's address. Upon reception of the heartbeat, the support service will validate the solution, provide an acknowledgement receipt that the peer's heartbeat will be included in the next block, and will propagate the heartbeat message to other peers. Heartbeats will be mined into block based on a majority vote that ensures that each block is final.

The heartbeat acknowledgement receipt should contain, among other things, the current header block hash and block number of the heartbeat registry blockchain. If the peer falls behind tracking that chain, it can realize that from the returned block number and request the missing blocks from some support service node during its next heartbeat exchange. In addition, during their own interactions, the mining peers can exchange information about their views on the heartbeat registry chain to detect potential anomaly and synchronize this chain directly.

If designed in this manner, the mining peers can individually decide their own course of actions for the checkpoint protocol and easily verify the veracity of the support service or any of its nodes. Finally, note that the support service nodes can be realized in many different manners in a PoW blockchain network. For example, they can be selected from the mining peer population based on some notion of deposited stakes [2]. Alternatively, some dedicated non-mining nodes with guaranteed pair-wise network connectivity can be elected to provide the service. The core concern in support service nodes selection is network connectivity – not trust – as they have little leverage in manipulating the checkpoint protocol.

[2]A peer can further confuse the support service's reasoning in this matter by seeking heartbeat acknowledgements for old blocks instead of its header block at random intervals if the header is mined by others.

B. Valid Voter Identification

Ensuring that only valid votes are counted is a prerequisite to any voting based distributed consensus protocol. Our checkpoint protocol is no exception. If precautions are not taken, a tricky front-runner can run dummy identities that would vote for it and bias the result in its favor, or a malicious attacker can confuse the support service in believing that consensus has been reached and force it to publish checkpoint block sealing materials for an invalid chain.

The simplest solution to this problem is to make the mining peers go through some *KYC* process and acquire public certificates to be able to participate in the checkpoint protocol. This is the approach taken in *KONA blockchain platform*. As an alternative, an approach similar to Ethereum's validator stake deposit [2] can be adopted for voter registration. The most flexible solution to the problem that does not compromise the anonymity of the mining peers is to make the heartbeat exchange puzzle difficult enough to dissuade fake voters. However, that would significantly increase the network maintenance cost. To summarize, what solution to voter identification is appropriate depends on the nature of the underlying blockchain network.

VIII. Conclusion

Transaction finality guarantee for blockchain ledger is important for blockchain technology's adoption in recording and enforcing real world financial and business matters where actions are seldom reversible. This paper presented and analyzed our transaction finality solution for the *KONA blockchain network:* a proof of work (PoW) mining based public blockchain network with smart contract support and authorized mining nodes. We provide transaction finality by establishing network-wide consensus about an irreversible ledger state – we call a *checkpoint* – at deterministic intervals. The consequent transaction finality protocol is called the *checkpoint protocol*.

We introduced a notion of a distributed support service in the blockchain network that is responsible for recording heartbeat messages of the active mining peers for population estimation and to track the progress of the checkpoint establishment process. A network-wide consensus about a checkpoint is reached by a 51% majority voting on a fair ledger election process by the active mining population and recorded through their heartbeat message exchanges with the support service. The nature of interactions with the support service ensures that mining peers know about the global state of the network within a bounded time and no broadcast is needed for a consensus establishment.

There is room for future improvement of our checkpoint protocol. Interested researchers can investigate how to reduce both the number of rounds and the timespan of individual rounds and still guarantee a convergence. The most challenging improvement will be to eliminate the support service nodes altogether and modify the PoW incentive of the mining nodes in such a manner that they can change their network connectivity and participation behavior at periodic intervals to establish checkpoints on their own.

References

[1] Bitcoin energy consumption index. https://digiconomist.net/bitcoin-energy-consumption. Accessed: 2019-02-14.

[2] Casper proof of stake compendium. https://github.com/ethereum/wiki/wiki/Casper-Proof-of-Stake-compendium. Accessed: 2019-02-14.

[3] Gossip protocol. https://en.wikipedia.org/wiki/Gossip_protocol. Accessed: 2019-03-02.

[4] How blockchains could change the world. https://www.mckinsey.com/industries/high-tech/our-insights/how-blockchains-could-change. Accessed: 2019-01-25.

[5] Top 100 cryptocurrencies by market capitalization. https://coinmarketcap.com/. Accessed: 2019-07-04.

[6] Adam Back. Hashcash - a denial of service counter-measure. Technical report, 2002.

[7] Michael Barborak, Anton Dahbura, and Miroslaw Malek. The consensus problem in fault-tolerant computing. *ACM Comput. Surv.*, 25(2):171–220, June 1993.

[8] Ethan Buchman, Jae Young Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *CoRR*, abs/1807.04938, 2018.

[9] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[10] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On security analysis of proof-of-elapsed-time (poet). pages 282–297, 10 2017.

[11] D. Stalin David. The ripple protocol consensus algorithm. 2014.

[12] Yun Ding, Patrick Horster, and Holger Petersen. *A new approach for delegation using hierarchical delegation tokens*, pages 128–143. Springer US, Boston, MA, 1996.

[13] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, January 1987.

[14] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 251–260, London, UK, UK, 2002. Springer-Verlag.

[15] Oğuzhan Ersoy, Zhijie Ren, Erkin Zekeriya, and Reginald L. Lagendijk. Transaction propagation on permissionless blockchains: Incentive and routing mechanisms. 06 2018.

[16] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *FCT*, 1983.

[17] Guerney D. H. Hunt and Lawrence Koved. Blockchain checkpoints and certified checkpoints, May 2018.

[18] Pankaj Jalote. *Fault Tolerance in Distributed Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[19] Joshua A. Kroll, Ian Davey, and Edward W. Felten. The economics of bitcoin mining , or bitcoin in the presence of adversaries. 2013.

[20] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[21] David Mazières. The stellar consensus protocol: A federated model for internet-level consensus, 2015.

[22] Fedor Muratov, Andrei Lebedev, Nikolai Iushkevich, Bulat Nasrulin, and Makoto Takemiya. Yac: Bft consensus algorithm for blockchain. 09 2018.

[23] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[24] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 643–673, Cham, 2017. Springer International Publishing.

[25] K. Saito and H. Yamada. What's so different about blockchain? — blockchain is a probabilistic state machine. In *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 168–175, June 2016.

[26] Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997.

[27] John Turek and Dennis Shasha. The many faces of consensus in distributed systems. *Computer*, 25(6):8–17, June 1992.

[28] D. Wood. Ethereum: a secure decentralised generalised transaction ledger. 2014.

[29] Shui Yu. *Distributed Denial of Service Attack and Defense*. Springer Publishing Company, Incorporated, 2013.