A Distributed Consensus Algorithm for Cryptocurrency Networks

by user johnstuartmill * and an anonymous user

October 22, 2016

Abstract

In this paper we present an Algorithm for Distributed Consensus. The object of consensus is a hash code of some data (e.g. of a candidate block for appending to a blockchain). The subjects of consensus are distributed nodes that attempt to agree on what the most correct hash code is. The objective of the authors of this paper is to find an optimal set of rules for each node to follow during consensus trials, so that the final agreement between nodes (i) can be reached fast, (ii) would require minimal network traffic, and (iii) can withstand a large-scale coordinated attack by a well-organized network of malicious nodes. By design, the algorithm is a scalable and computationally-inexpensive alternative to proof-of-work, therefore both the consensus algorithm and block-making can run on a budget hardware that have low price and low energy consumption, thus making the cryptocurrency network robust to possible centralization attempts.

Contents

*Funded by [REDACTED].

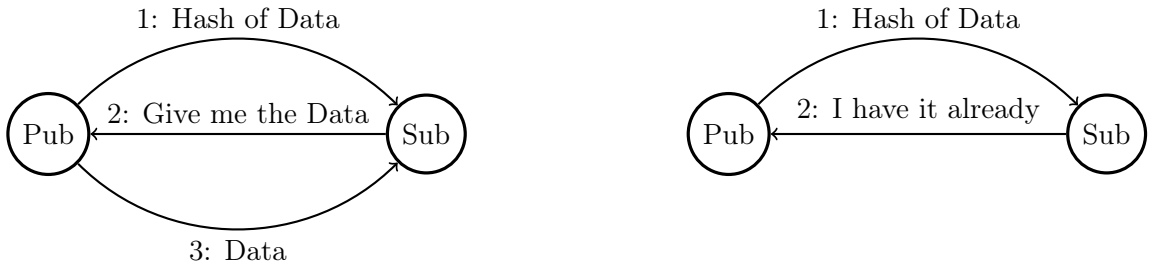

Figure 1: Data notification and data request. On the diagram to the left, the subscriber ("Sub") has not seen the hash, so the corresponding data was requested from the publisher ("Pub"), then delivered. On the diagram to the right, the hash has already been seen by the subscriber, so the corresponding data was not requested.

0.1 Preface

This paper is organized as follows. First, in Section 0.2, we give a brief introduction of mesh network. Then, in Section 0.3 we describe the algorithm. Simulation results are presented in Section 0.4. Finally, Section 0.5 show pseudo-code of the algorithm.

The reader is expected to have a basic understanding of (i) the concept of hash function, (ii) how public/private cryptographic keys are used to sign hashes and verify signatures. The concepts of block and blockchain are explained in the text.

0.2 Mesh Network

0.2.1 Sparse Connectivity

Mesh Network is designed for sparse connectivity. Each network node ("node") is expected to be directly connected to only a few upstream nodes and a few downstream nodes[1]. Each node plays a dual role: it is both a publisher and a subscriber. As a subscriber, it receives data from the upstream nodes it is directly connected to ("publishers"). As a publisher, the node (i) forwards data it has received and (ii) sends the data it generated independently to the downstream nodes it is directly connected to ("subscribers"). Such design allows a node to receive data from potentially each and every node, and to send data to potentially each and every node on the network. See Fig. 2 for an example of physical versus logical connectivity; Fig. 1 illustrates data propagation.

0.2.2 Node's Privacy

The nodes are addressed by their cryptographic public key ("Pubkey"). Node's IP address is only known to the nodes to which it is connected directly.

0.2.3 Message Propagation and Public Broadcast

Any node can publish a message. The downstream nodes that received a message forward it, after a cryptographic validation and checking that the message is not a duplicate, to their respective downstream nodes. The result is broadcasting the message to the entire network.

[1] Additional connections create routes with less hops, thus speeding up message propagation.

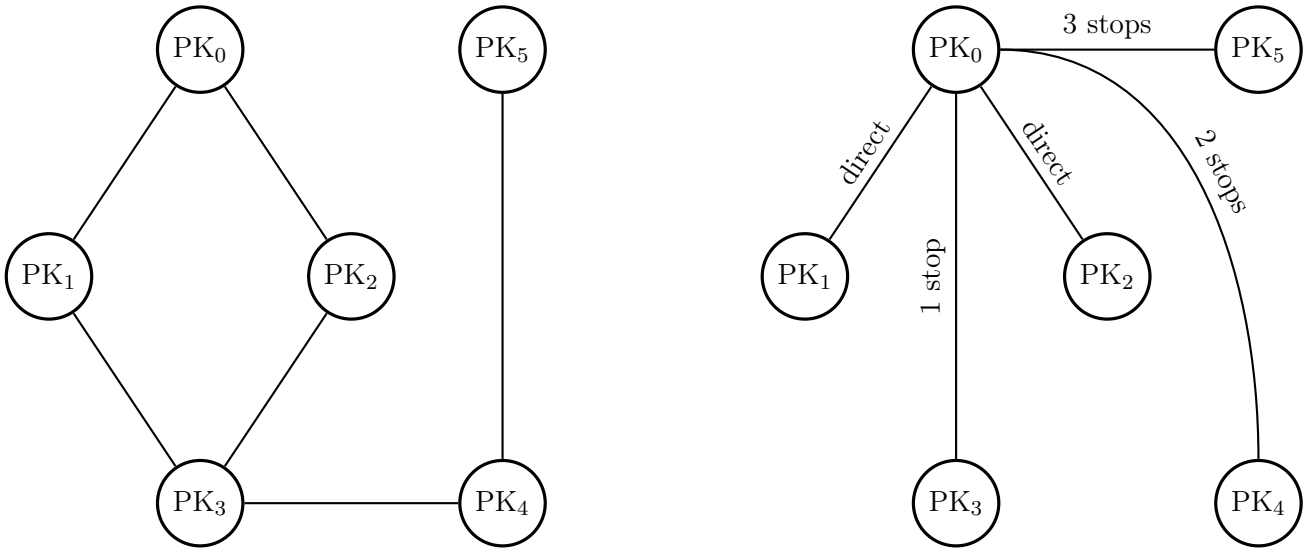

Figure 2: Physical connectivity of the nodes in a small Mesh Network (left), and logical connectivity of *node* PK_0 *only* (right). The "PK" on the diagram stands for "public key".

0.2.4 Controlling Data Content

By Disconnecting

A node can disconnect from a node directly connected to it.

Likely use-cases: (i) node Y is forwarding traffic at a rate that node X can not handle; (ii) node Y is forwarding messages with a large fraction of inappropriate or malicious content.

Other nodes in the network can still receive messages from Y if there exist a route to Y that does not involve X.

By Blacklisting

A node can locally blacklist any other node by Pubkey.

After node X blacklists node Y, the messages coming from Y are ignored by X, and are not forwarded to X's subscribers.

A likely use-case: node Y is forwarding messages with a large fraction of inappropriate or malicious content, but X cannot disconnect from Y because Y is not directly connect to X.

Other nodes in the network can still receive messages from Y if there exist a route to Y that does not involve X.

0.3 Network Consensus Algorithm

0.3.1 Need for Consensus

One of the requirements for a well-functioning cryptocurrency is that all network participants have access to the same list of transactions[2], both historical and recent. Achieving this without centralization requires a special mechanism. One such mechanism is Network Consensus Algorithm described below.

[2]In practice, the list of transactions is divided into groups called *blocks*, and the blocks are connected into what is called *blockchain*. "Bagging" transactions into blocks and then linking blocks into a chain is done in such a way that it is computationally unfeasible to commit fraud by modifying historic transactions stored within blocks.

The primary purpose of the consensus algorithm is to synchronize the state of the blockchain across all the network nodes. One consequence of having a synchronized blockchain is consistent accounting, i.e. calculation of coin balance for a given Pubkey yields same value at each node that performed the calculation.

0.3.2 Designing the Algorithm

We experimented with some consensus algorithms in which nodes are averse to having an opinion that differs from that of (a) their nearest neighbors or (b) a small set of local neighbors. We found, in simulation, that such behavioral model allows the consensus to be easily manipulated by a relatively small fraction of malicious nodes. The effect is especially strong in the presence of cycles in the (directed) connectivity graph[3]. We dropped such models from further consideration as we realized that they have a fundamental flaw: the behavior of the nodes are being *modeled after unintelligent individuals*. Not surprisingly, a small organized intelligent group can change herd's opinion in an arbitrary way.

Another class of algorithms, that we also rejected, involve electing a leader node. Agreeing to elect one's leader (or a temporary ruler), we contend, is not a very intelligent behavior either. Here is why. Leader election is a natural adaptation in situations when group's survival requires high intelligence, while the average intelligence of group members is low[4]. Hence the group, in order to to survive, has to find a member who can make intelligent decisions for the group. Such behavior is *modeled after sheeple*, or after species that have a predilection for being led, which does not seem to be congruent with cryptocurrency community. Additionally, a leader is potentially a single point of failure, as she can be coerced by a malicious entity to act against the interests of the society she was elected to represent. Therefore we decided to drop leader-based models from consideration as well.

The above restrictions lead to the following requirements for the nodes:

- Node is *intelligent*. Each node is able to form its own independent opinion (e.g. best next block) by doing a robust statistical analysis of the opinions it received.
- Node is *skeptical*. Each node always performs authorship verification and fraud detection.
- Node is *sovereign*. While other node's opinions are taken into account, the node neither align itself with any group or authority, nor it seeks a payment in return for supporting a given opinion.
- Node is a *content generator*. The node is able to receive raw data (e.g. low-level, elementary events such as transactions) and produce an independent research that leads to a new opinion (e.g. block hash).

The above requirements lend itself to an algorithmic implementation. See Section 0.3.7 for detail.

0.3.3 Scalability With Network Size

As it was mentioned in Section 0.2.1, the node can potentially receive broadcast messages from any node in the network, even if the node is directly connected to only a few other nodes. As the network size grows, so does the number of consensus-related messages. A large number of data-points (e,g. opinions) is beneficial for obtaining statistically-significant results. However, a large data sample that the node might want to acquire in order to make a high-quality consensus-related decision would, among other complications, involve waiting for the messages traveling along multi-hop routes in the Mesh Network, which, in turn, would put certain constraints on block-making rules. We avoid this and other related issues by limiting the sample size to a prescribed number. This parameter is denoted Z in the text below. The consequence of such decision is an implicit introduction of a node-specific *influence region* which consist of the nodes that happen to be among the closest, in terms of network latency, Z nodes to the node in question[5].

[3]Restricting one's analysis to acyclic graphs seems too restrictive: even the totalitarian societies tend to have cycles in their communication networks.

[4]We are curious if anthropologists or ethnographers could confirm this proposition, perhaps using their yet-unpublished research.

[5]The influence region, and its boundaries, fluctuates with the inter-node latency.

We explored the effect of Z on the results of consensus trials when the network is under a subversion attack. We found that Z does not have to be large: values of about 100 are acceptable. See Section 0.4.4 and Fig. 6 for detail.

In live deployment, we expect each node to select a node-specific value Z from a recommended range, so the the performance of the node as a consensus participant is maximized.

We also explored the effect of increasing the number of nodes, N, in the network, while keeping the number of block-making nodes, B, constant. We found that the quality of consensus results are not affected by the increase. Please see Section 0.4.3 and Fig. 5 for detail.

The effect of network size on message propagation times was not explored here and is a subject of a follow-up paper.

0.3.4 Before Consensus Trials

Before participating in consensus trials, the node might want, as a part of initialization, to download the tail of the blockchain (i.e. the most recent n blocks).

During normal operations, the node might want to check, from time to time, how many nodes in the network are in agreement with the node's decisions on blocks.

These two related tasks are performed by a separate algorithm.

0.3.5 Finalizing Consensus Trials

Consensus trials for more than one block sequence number can be open simultaneously. This is analogous to a criminologist collecting evidence simultaneously for more than one case: when enough data is collected for one of the open cases *and* a conclusion on the case can be reached, the case is closed.

When the node has acquired Z opinions for a given block sequence number, or when block clock has advanced much beyond the sequence number, the node closes the trial and calculates the winning candidate hash code.

Recall that during consensus trials the nodes exchange small messages that contain block hash code. A (much) larger message containing the block itself will be requested from the network (see Fig.1) after the node determines the winner. Deferring the block delivery until the winner is determined reduces network traffic. A possibility of non-delivery is already taken into account by the Algorithm.

The winning block is added to the local blockchain.

0.3.6 Independence Of Clock Synchronization

The Algorithm does not use "wall clock" (i.e. calendar date/time). Instead, block sequence numbers that are extracted from validated consensus- and blockchain- related messages are used to calculate node's internal time. This can be informally called "block clock".

0.3.7 Implementing Consensus Algorithm

Implementing the Algorithm requires to designate some nodes as *block-makers*. Potentially, every node can be a block-maker. A block-maker node (i) observes transactions published by other nodes, (ii) combines, according to a prescribed, deterministic set of rules, the observed transactions together into a block called *candidate block*, and (iii) broadcasts to the network a message containing the following four items:

- hash code of the candidate block,
- signature of the hash code (done with node's private key),
- node's public key, and
- block's sequence number.

The candidate block itself is broadcast, upon request, as a separate message.

The purpose of making candidate blocks by multiple nodes is to *generate a large number of independent, observation-based, cryptographically-signed opinions*. Specifically, if all block-makers (i) started with identical copies of blockchain, (ii) received same set of transactions, and (iii) followed the prescribed

block-making rules, then the candidate blocks they produced would be identical. In a realistic networking environment some nodes would not receive all the messages published so far, therefore the candidate blocks they published would be different from candidate blocks published by fully-informed nodes. Nevertheless, when message arrival rate is not too high for the inter-node latency (in terms of probability distribution), practice shows that majority of block-making nodes can build identical candidate blocks for a given sequence number, therefore they would broadcast identical hash code. A high redundancy of the correct candidate hash code provides a protection against large-scale coordinated attempts to defraud the network in which the attacking entity sets up a large number of malicious nodes that broadcast a hash code of a chosen fraudulent candidate block.

All nodes listen to the broadcast of hash codes of candidate blocks, and *build a data sample of opinions* from which they attempt to make statistical inference as to what the correct hash code could be.

The nodes with a poor network connection would not be able to observe all the transactions, therefore they should not be designated as block-makers[6]. Nevertheless, such nodes can still receive messages with hash codes of candidate blocks; these messages are sent at much lower frequency than transactions, and they contain only several items. Therefore the non-block-making nodes can still maintain their own local copy of up-to-date blockchain.

Our simulation results show that the breakdown coefficient of the Consensus Algorithm presented in this paper is between 0.4 and 0.5, that is, the adversary would need to have that large a fraction of malicious block-makers in order to get a desired fraudulent hash to win the consensus. We refer the reader to Section 0.4.4 for further simulation results.

The consensus is not guaranteed to be achieved by each and every node. For example, when a node (or a cluster of nodes) becomes disconnected from the rest of the network, its blockchain becomes stale as soon as the rest of the network generates a new block. As another example, consider a honest node physically connected to only malicious nodes. The malicious nodes discard incoming network broadcast, and, instead, send malicious data to the surrounded node. End example. When a publisher is suspected to be malicious, the subscriber can blacklist its Pubkey and attempt to re-synchronize it local blockchain with the network.

0.3.8 Consensus Algorithm By Example

The intuition behind the consensus algorithm can be demonstrated by an example that focuses on one node. The node collects consensus-related messages, detects some fraudulent attempts, and proceeds to determining the winning hash code (hence, the winning block). Fig. 3 shows data flow, while Table 1 shows data analysis. The captions provide detailed descriptions.

0.4 Simulation

0.4.1 Network Parameters

Consider a network of N nodes. Each node can have upstream and downstream connections. On average, each node is directly connected to S downstream nodes, thus making the average number of upstream connections also S. The network has B block-makers, with $B \leq N$. Each node determines the winner after receiving candidate hashes from Z block-makers.

In general, there is no restriction on connection topology. We simulated two networks of varying sizes, described below.

In the first network, each node was connected randomly to S publishers, so each automatically acquired S subscribers on average. It models a modern city with either a single ISP or multiple ISPs who happen not to de-prioritize requests coming from other IPSs. This network is a base case for testing the Algorithm.

In the second network the nodes are placed on a *circle*, then connected to the nearest S publishers, and acquired S (nearest) subscribers on average. Please note that in spite of being directly connect to only nearest neighbors, the nodes receive messages from remote nodes as well due to forwarding feature

[6]The chance of their candidate hash winning the consensus would be low due to a high chance of missing some transactions, therefore broadcasting candidate hash would likely to be a waste of network bandwidth.

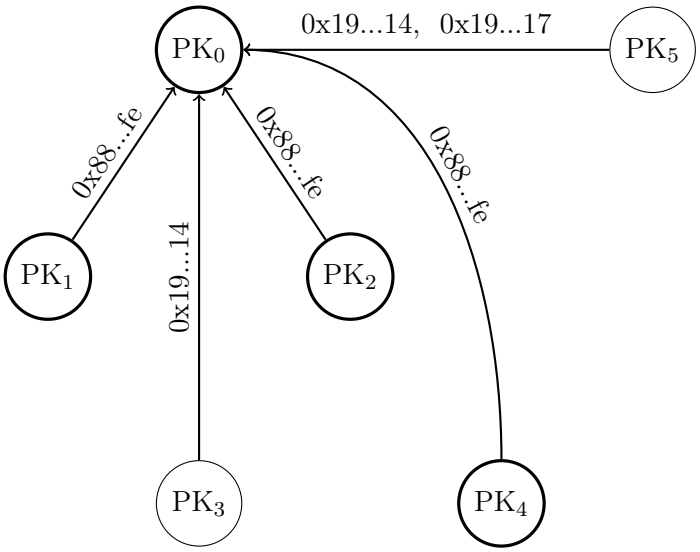

Figure 3: Consensus data flow as seen by node PK_0 in the network depicted previously in Fig. 2. All messages that are sent to PK_0, both direct and forwarded, are shown. Vertices connect block-making nodes to node PK_0. Vertices are labeled with the hash code being sent. Arrows indicate the direction of data flow. Given the (i) state of blockchain, (ii) observed transactions, and (iii) the block-making rules, the hash of the correct candidate block should be 0x88...fe. The nodes PK_3 and PK_5 have conspired to insert a fraudulent transaction into the blockchain, and both are proposing a block whose hash code is 0x19...14. Additionally, the PK_5 is attempting to send hash code 0x19...17 for the same block sequence number, in hopes of decreasing statistical significance of the correct candidate. See Table 1 for determining the winning hash code.

Hash code of candidate block	List of *unique* Pubkeys that sent that hash code	List length as a proxy for a number of independent opinions	Is a winner of current consensus trial?	Comments
`0x19...14`	PK_3 PK_5	2	No	
`0x19...17`		0	No	PK_5 is banned for spam
`0x88...fe`	PK_1 PK_2 PK_4	3	Yes	

Table 1: Determination of the winner for the consensus depicted in Fig. 3. For a given block sequence number of a future block, the Algorithm maps a hash code (column 1) to a set of Pubkeys (column 2) that sent the hash code. The Algorithm decides, deterministically and independently on each node, when to stop accepting new data so that the winning hash code can be determined. Column 3 shows the number of Pubkeys in each set, and the winner is a hash that has the largest number. Ties are resolved deterministically. Column 4 shows if a given hash is the winner. In this particular example, the 0x19...14 was received from two Pubkeys, and the 0x88...fe was received from three Pubkeys. The latter, 0x88...fe, won 60% of votes, more than other candidates. According to the Algorithm, the code 0x19...17 was determined to be a malicious attempt, therefore it was excluded from the contest and its sender, Pubkey PK_5, was banned for a period of time. We showed propagation of 0x19...17 from PK_5 to PK_0 to illustrate hash validation; the 0x19...17 is expected to be intercepted and discarded by the first correctly functioning node, i.e. PK_4.

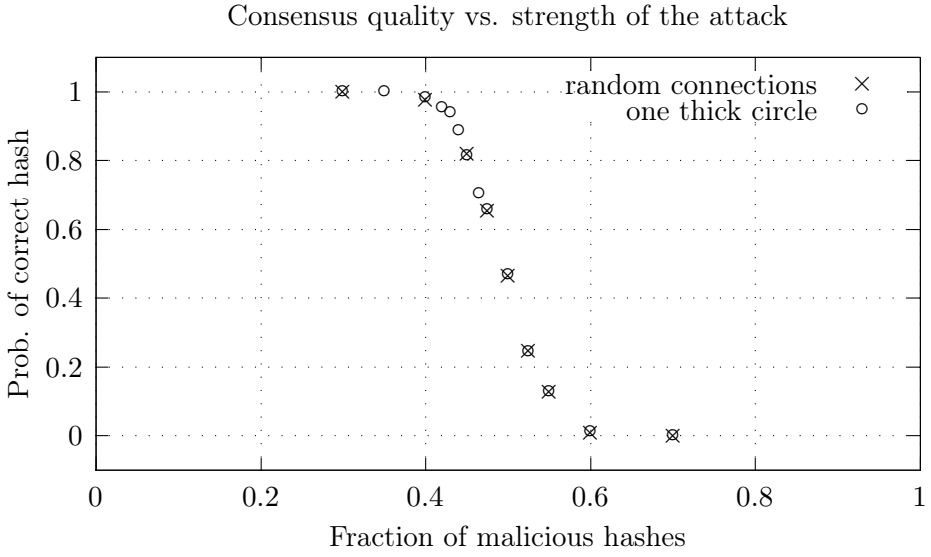

Figure 4: The effect of network topology on consensus quality. The Figure shows the probability of obtaining the correct hash (vertical axis) versus fraction of malicious nodes (horizontal axis) for two networks. The networks have N=10000, B=1000, S=5, Z=100 parameters, but differ in connectivity graph. The key reason for a nearly-identical response to attacks is the forwarding feature of Mesh Network. See Section 0.4.2 for a discussion.

of Mesh Networks. This network topology was chosen so that we can easily detect non-convergence of opinions, if this is to happen[7].

In both cases we used a uniformly-distributed[8] probability distribution of inter-node latency between 100 and 400 milliseconds. The cryptographic calculations required for making and verifying signatures were made part of the simulation.

0.4.2 The Effect Of Network Topology

In order to assess the effect of network topology on the consensus, we simulated the two networks using N=10000, B=1000, S=5, Z=100 parameters. Fig. 4 shows that for the given parameters N, B, S and Z, the effect is negligible, in spite of big difference in the connectivity graph. The reason for the nearly-identical response to the attack is the forwarding feature of Mesh Network: receiving messages from another node does not require a direct connection. Consequently, if the network's ability to withstand a subversion attack is not sensitive to connectivity graph, then each node can directly connect only to the Pubkeys it trusts, without a fear that this might create an unusual connectivity graph that is susceptible to an attack.

This result is encouraging for two reasons. First, it suggest that Mesh Network has been, so far, a good choice for deploying cryptocurrency in a hostile environment. Second, network connectivity graph, as a technical parameter, can be dropped from some (but not all) theoretical analyses.

While the effect on consensus is very small, the effect on time required by nodes to collect enough candidate hashes is not: the circular network took more forwarding and hence more time.

[7]Non-convergence can be illustrated with the following example. Place nodes in a circle, and connect them as follows: each node's next neighbor clockwise is its only subscriber, and the previous neighbor is its only publisher. Choose a consensus algorithm in which the node joins/copies the opinion of its publisher. The opinions are limited to two values, say, 0 and 1. Give the nodes arbitrary initial opinions with some 0s and some 1s. Then run the consensus trial while updating nodes opinions synchronously. One can easily see that the circular sequence of 0s and 1s would be rotating clockwise, with the nodes changing their opinions and never reaching a stable-state.

[8]We expect to explore the effect of latency distribution on reaching the consensus in a follow-up paper.

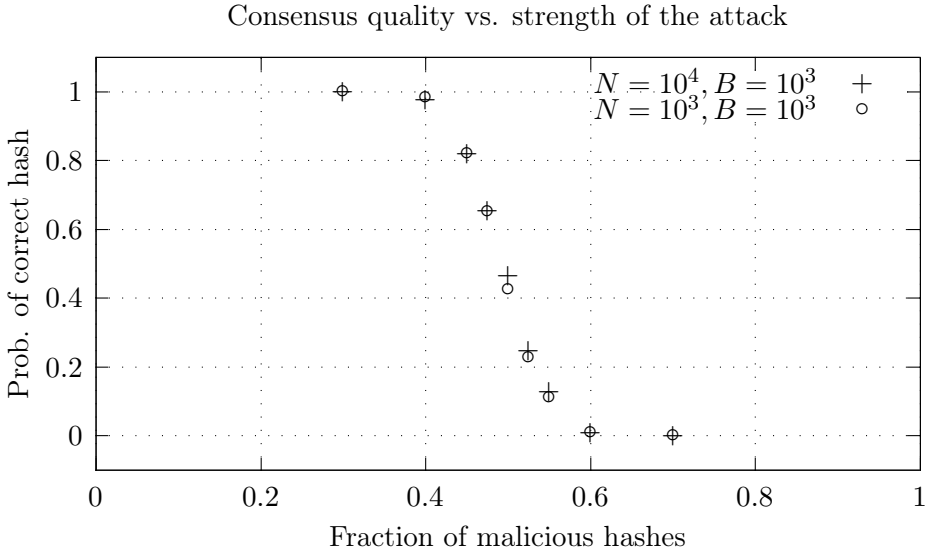

Figure 5: The effect of passive nodes on consensus quality. The two networks have unstructured random connection graph, B=1000, S=5, Z=100 and $N \in \{1000, 10000\}$. Note that the network parameters were identical in both cases, except the latter one had 900 non-block-makers. The figure shows that the outcome of the consensus is not influenced by passive, or non-block-making, nodes.

0.4.3 The Effect Of Passive Nodes

To see the effect of non-block-makers on consensus, we simulated two networks using unstructured random connection graph, B=1000, S=5, Z=100 and $N \in \{1000, 10000\}$. Note that the network parameters were identical in both cases, except the latter one had 900 non-block-makers. The result is shown on Fig. 5. Apparently, the outcome of the consensus is not influenced by passive nodes. Instead, it is determined by the ratio of honest nodes to malicious nodes.

We find this result encouraging as well: it suggest that adding passive nodes (the ones that that transact coins and forward messages but do not make blocks themselves) would not degrade network's resilience to attacks. This, in turn, suggests that users with low-performance low-cost hardware would not be a drag.

0.4.4 The Effect Of Data Sample Size

To explore the effect of sample size Z on the consensus, we simulated a network with circular connectivity graph, N=1000, B=1000, S=5, and $Z \in \{25, 100, 200, 1000\}$.

To facilitate interpreting the results, let us say that a consensus trial completed successfully when two conditions were met: (i) at least 80% of nodes agreed on a hash code, and (ii) the agreed-upon hash code was not the fraudulent one. Also, let us call the maximal fraction of malicious nodes (or malicious hashes sent) that still allow for a successful completion the *breakdown coefficient*. The larger the coefficient, the more robust the network is with respect to a subversion attack.

The results presented in Fig. 6 show that when each node makes a decision after collecting as little as Z=25 candidate hashes, the consensus is still a success in spite of having 40% of candidate messages that have been manufactured to subvert the consensus. In this case the breakdown coefficient is 0.4. When Z increases to 100, the breakdown coefficient increases to 0.45. For a much larger sample of Z=1000, the breakdown coefficient is close to 0.5; nearly all the nodes agree on the correct hash, for as long as the fraction of fraudulent candidates is below 50%.

We conclude that the Algorithm is sufficiently robust with respect to the specified kind of attack.

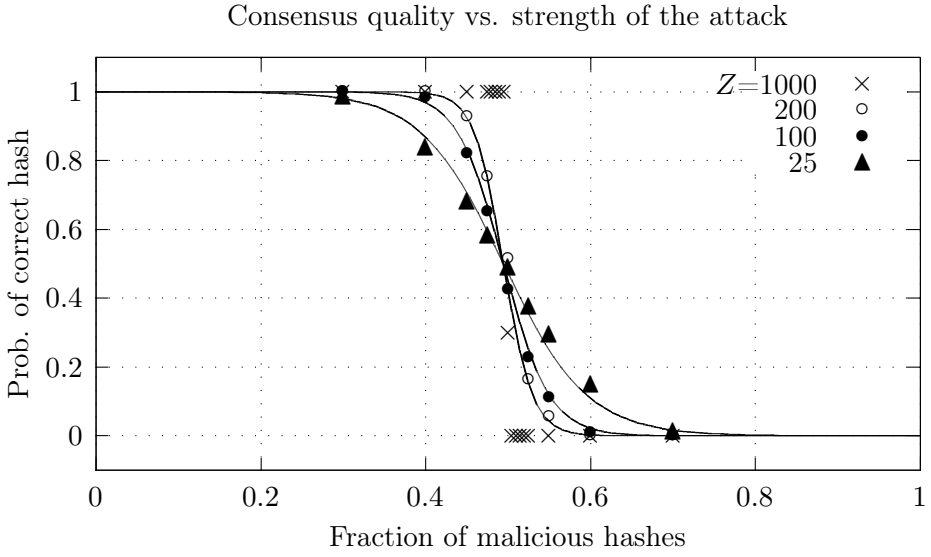

Figure 6: Probability of obtaining the correct hash (vertical axis) versus fraction of malicious nodes (horizontal axis). The varying parameter, Z, is the number of data-points used by nodes use for decision-making. Malicious nodes perform a coordinated attack in which they broadcast an identical fraudulent hash code. The figure shows that increasing the size of decision-making sample, Z, increases the robustness of consensus trials. The network has circular connectivity graph, N=1000, B=1000, S=5, and $Z \in \{25, 100, 200, 1000\}$.

0.5 Algorithm Pseudo-Code

Algorithm's pseudo-code is shown in Fig. 7, and is self-explanatory. See the caption for detail.

0.6 Acknowledgments

The authors are thankful to several members of altcoin community, who wish to remain anonymous, for fruitful discussions.

0.7 Legal Notice

```
handle_consensus_data(pubkey,hash,sig,seqno) {

  if (!(is_consensus_data_valid(pubkey,hash,sig,seqno)))
    // Ignore data. Consider banning 'pubkey' for failure to follow data specification.
    return;

  if (!(verify_cryptographic_signature(pubkey,hash,sig)))
    // Someone is trying to impersonate 'pubkey', so ignore the data.
    return;

  if (!is_seqno_relevant(seqno, blockchain))
    // Either trials completed for this seqno, or the seqno is too large.
    return;

  // Get consensus data we collected so far for the given block sequence number:
  ConsensusInfoType& info = db[seqno];
  if (info.is_consensus_closed())
    return;

  if (info.Z.contains(pubkey))
    // Public key 'pubkey' has already sent its opinion. This could be a malicious attempt.
    // Ignore data. Consider banning 'pubkey' for failure to follow consensus protocol.
    return;

  info.X[pubkey] = ConsensusDatum(pubkey,hash,sig,seqno);
  info.Y[hash].insert(pubkey);       // This maps hash --> set of unique pubkeys
  info.Z.insert(pubkey);             // These are unique pubkeys

  if (info.Z.length() < cfg_max_sample_size)
    return; // Sample size is too small to infer population parameters.

  // At this point we have collected the amount of data that we think is sufficient to
  // make statistical inference. The following steps are verbose when expressed
  // programmatically, so we describe them in words:

  // 1. For each key in info.Y map, extract data into a list of pairs
  //    (key, info.Y[key].size()).
  // 2. Sort the list according to the 2nd element, resolve any ties using the 1st.
  // 3. The 1st element of the last record in the sorted list is the hash code that has
  //    the largest number of unique pubkeys. Call it best_hash.
  // 4. Get the data that corresponds to best_hash. The 'info.X' is used here.
  // 5. Prepare the data to be appended to the local blockchain.

  db.erase(seqno); // No longer needed.
  return;
}
```

Figure 7: Pseudo-code of the Consensus Algorithm. For clarity of exposition, it is stripped of some features such as error handling and fraud detection. The complete version of the Algorithm can be found in https://github.com/johnstuartmill/consensus