Sia: Decentralized, Compensated, Self-Repairing Computer Storage

David Vorick
Nebulous Labs

Luke Champine
Nebulous Labs

September 13, 2014

Abstract

Sia is a decentralized platform for computer storage. It operates as a marketplace, where storage providers offer up their disk space to be rented by anyone with storage needs. The price of storage is uniform across the network, and is adjusted in real-time by a market making algorithm. The medium of exchange is a new cryptocurrency, Siacoin. Files on Sia are distributed to a large number of randomly selected hosts. This allows for massively parallel downloads, resulting in high throughput. Erasure coding is used to provide redundancy, enabling highly resilient storage at lower redundancy than standard backups. These factors make Sia a superior alternative to centralized cloud storage offerings.

1 Introduction

Sia is a decentralized platform for storing data on the cloud. Specifically, Sia provides block level storage, like an unformatted hard drive, making it filesystem agnostic. This means that it does not make much sense to talk about storage in terms of files. Instead, the term *sector* is used, a sector being a logical set of bits. Files can be stored within sectors or across multiple sectors. Sia is random access, meaning any part of a sector can be downloaded individually without needed to download the whole sector.

Each sector is spread across 128 hosts that monitor the health of the sector. If a host goes offline, it is automatically replaced by the network. Redundancy keeps the sector available throughout the replacement process. The sector is broken into 128 *segments*, of which m are redundant, where m is chosen by the uploader. Any m hosts can go offline simultaneously and the sector will still be recoverable. This means that a relatively small value of m is sufficient to protect sectors against catastrophic disaster scenarios.

The erasure code used is Reed-Solomon, which is a maximum-distance separable code. This means that redundancy is maximally efficient. For example, take a 100 MB sector that is encoded as 128 pieces with $m = 28$. 100 pieces are non-redundant, and 28 are redundant. Of these pieces, any 28 pieces can be lost and the full sector can still be recovered. Together, they consume 128 MB of physical storage, for a redundancy of 1.28. The default value for m is 28.

Sectors can be downloaded in parallel, since each segment of a sector is stored on a separate host. This allows for very high throughput, even if each host is only offering a little bandwidth. Latency can be reduced by contacting only the closest $128 - m$ hosts.

Each sector stored on Sia is accompanied by approximately 40 KB of overhead that allows the network to monitor and error-check the sector. Multiple files can be stored on a single sector; 40 KB is not a minimum file size, simply a minimum sector size. The maximum size of a segment is limited to 1 MB. The maximum size of a sector therefore depends on the value of m: a larger m means a smaller maximum sector size. Optimizing for smaller sectors means that each file is stored across more hosts, which improves resilience and download speed.

Another property of Sia is that it makes censorship very difficult. Forcefully taking sectors off of the network requires corrupting or destroying more than m of the 128 segments that compose the sector. Be-

cause sectors on Sia are randomly distributed among a global set of hosts, it is very difficult for a single entity to attack a sufficient number of machines to corrupt a segment, especially when m is large.

2 Managing Sectors

Sectors are encrypted by default, but this is purely optional. Unencrypted sectors are convenient for content that is intended to be shared, such as images and videos.

The contents of a sector can be altered at any time, but only by authorized parties. Authorization is handled via public key cryptography. It is also possible to create an immutable sector, one that cannot be altered after it has been uploaded. This is achieved by authorizing nobody, including the uploader. Immutable sectors are appropriate for static content such as media, where there is an expectation that the data will not change.

Each sector is associated with a balance of siacoins. Siacoins are periodically subtracted from this balance and divided equally among the sector's hosts. When the balance reaches zero, the sector is marked for deletion. Anybody can add to the balance of a sector at any time. It should be noted that immutable sectors will only be removed from the network when their balance fully depletes.

3 Economics

Sia compensates hosts using a cryptocurrency. This cryptocurrency, Siacoin, will be easily exchangeable for bitcoins and subsequently USD through a trustless exchange. When the currency launches, the mining rate will be set to 10,000 siacoins per day. The number of coins mined will decrease daily until the 4 year mark, at which point the network will mine coins such that the annual inflation rate is kept permanently at 5%. This means that there is no cap to the number of siacoins that will be produced. No coins will be premined.

An inflationary model has been chosen for Siacoin to shift its economic incentives from early adopters to current hosts. Initially, the number of coins generated will be very high, and early adopters will be rewarded for joining the network in its infancy. In the long run, however, those actively contributing resources to the network should be the most heavily rewarded.

It has not been determined how newly mined coins will be distributed. They cannot be given to the hosts on the network, because this creates a recursive storage attack that would negatively impact performance on the network and allow attackers to mine coins they do not deserve. The reward for the hosts is instead payment from the people renting the storage.

To fund development and reward the early backers of Sia, we introduce a new concept called Siastock. Siastock is a 100% premined currency that will be initially owned solely by the developers. During an IPO or a pre-IPO, some percentage of Siastock will be traded for seed money. All income from hosting transactions is taxed 3.9%, which is sent to the owners of Siastock. Someone who owns 1% of all Siastock will receive $1\% * 3.9\% = 0.039\%$ of all host income on the Sia network. This model provides an avenue for investors and speculators to profit without negatively impacting the primary medium of exchange, Siacoin.

The storage on Sia has a certain value, and the only way to access that storage is to spend siacoins. If the price of Siacoin drops significantly, access to Sia resources becomes very cheap, which should in turn increase the demand for Siacoin and bring the price back up. The value of storage, combined with the fact that this value can only be accessed using Siacoin, gives Siacoin intrinsic value proportional to the resource it makes available. Siastock provides a guaranteed income that is proportional to total network activity, so Siastock also has intrinsic value.

4 Quorums

When a host joins the network, it joins as a set of participants. Each participant contributes exactly 8 GB of storage to the network, and is assigned to a random *quorum* on the network. A quorum is a set of 128 participants acting in consensus. Each quorum maintains a *state*, which is a set of metadata relevant to the quorum. A state contains a list of participants,

a list of sectors being stored, a list of wallets and balances, etc. The state is updated in *blocks*. Each block contains an update, called a *heartbeat*, from every participant. Heartbeats contain information that enables the quorum to:

- Confirm that each participant is still online
- Perform proof-of-storage for each participant
- Confirm transactions
- Prevent dishonest activity

Achieving consensus requires a solution to the Byzantine Generals Problem. Sia uses the solution with signed messages that is presented in the paper "The Byzantine Generals Problem", with slight variations. For each block:

1. All participants send their heartbeat to all other participants, with a signature. This creates a *signed heartbeat* - a heartbeat followed by a list of signatures from everyone who has seen it.
2. Each time a participant receives a signed heartbeat that they have not seen before, they add their signature to the list of signatures and send it to all other participants.

To ensure termination, time constraints are added, and heartbeats are sent around in steps. Each step lasts 15 seconds. All signed heartbeats received must have a number of signatures equivalent to the current step. Each signature must be from a different participant in the quorum. For example, during step 1, a heartbeat only needs to be signed by the participant producing the heartbeat. It is permissible for a heartbeat to be received containing more signatures than required. There are 128 steps total, the same as the number of participants. If a participant signs two different heartbeats from itself, that participant is considered dishonest and will be thrown from the network. After the final step, all honest participants will have the same information and will be able to compile the same block. The step is then reset to 1 and participants repeat the process for the next block.

The full algorithm:

1. The step counter is set to 1, and increases by 1 every 15 seconds.
2. Each participant creates a heartbeat, signs it, and sends it to every other participant in the quorum.
3. Received heartbeats are ignored if:
   (a) the number of signatures on the signed heartbeat is less than the step counter
   (b) there are multiple signatures from the same participant
   (c) there are signatures from non-participants
   (d) the heartbeat has been seen before
   (e) two other heartbeats from this sender have already been seen
4. For each heartbeat received, the recipient adds its signature and sends it to all other participants in the quorum.
5. When the step counter reaches 128, the heartbeats are compiled into a block.

The goal of each honest participant is to receive the same set of heartbeats as every other honest participant. If you are the first honest participant to receive a heartbeat, but it has sufficient signatures compared to the step counter, then you can be sure that every other honest participant will get the heartbeat because you will send it to them. If you are an honest participant receiving a heartbeat for the first time in the last step, you can be sure that all other honest participants have received the heartbeat, because there will be a signature from every other participant. There is one exception: the scenario in which a dishonest participant shares more than two heartbeats. In this case, every honest participant is guaranteed to receive at least two of the heartbeats. They might not receive the others because honest participants stop sharing after they have shared two. However, the honest participants will still throw the dishonest participant from the network, regardless of the contents or quantity of the heartbeats it has sent.

5 Block Compilation

Once step 128 is reached, the heartbeats are aggregated and block compilation occurs. All participants who submitted multiple heartbeats are penalized and thrown from the quorum, and their heartbeats are discarded. At this stage, the proof-of-storage algorithm is run. It extracts data from each heartbeat and verifies that the participant is honest about their storage. Each participant must also submit information relating to random number generation, which is parsed and verified as honest. Any participant that fails the storage proofs and/or random number proofs is thrown from the network and their heartbeats are discarded.

Next, transactions are processed. Before proceeding, the honest heartbeats are shuffled into a random order. The shuffling algorithm uses a deterministic seed from the previous block, such that all participants will arrive at the same random ordering. Transactions are then extracted from the heartbeats and processed one at a time. If a transaction is invalid, or is a double spend, that transaction is ignored. All participants are assumed to start with the same state, receive the same heartbeats, and process those heartbeats in the same order. Therefore, all participants will end with the same new state.

6 Proof-of-Storage

Each participant is responsible for storing 8 GB. This 8 GB is divided into many segments, each of maximum length 1 MB. Every segment stored by a participant corresponds to an equal-sized segment on every other participant in the quorum. Together, these corresponding segments form a ring. Each block, one ring is randomly chosen, weighted according to size, from the quorum to be uploaded to the network.

Each participant has the hash of every segment that forms the ring. Each participant must upload their entire segment, which can then be hashed and verified. This means that each block, up to 128 segments of size 1 MB may be uploaded in the block. Blocks are not stored permanently, so this does not create a bloated blockchain.

If a participant is not storing anything (or is only storing the hashes), the proof-of-storage algorithm is guaranteed to detect their dishonesty. If a participant is storing only half of the data, each storage proof carries a 50% chance of detecting the missing data. Though there are no guarantees that all corrupted data will be discovered, there is a strong guarantee that if a participant can pass the proof-of-storage test many blocks in a row, then the participant is storing almost all of the data.

This algorithm only works if the participants are already storing data. For this reason, a participant can only be paid for the storage that is in use, and can only mine on the storage that is in use. A pricing algorithm will ensure that this is most of the storage for all participants. A load balancing algorithm will ensure that all quorums have approximately the same percentage of free storage.

A dishonest participant could wait until it received enough heartbeats during step 1 to rebuild the original file and figure out what its own segment is supposed to look like. To prevent this, storage proofs take two blocks. In the first block, each participant must make a commitment, such that they are locked into producing only a single set of bits as their segment, but such that they cannot use the commitments from other participants to rebuild what their segment and commitment should look like. This commitment is produced by prepending a random string to the segment and then hashing the result. This random string is generated by the previous block and is the same for all participants. The random string prevents dishonest participants from being able to store what the commitment is supposed to look like, and forces participants to store the entire segment if they are to be able to produce a correct commitment. In the second block, the participants upload their segment. Their commitment can then be verified, and their segment can be verified.

These verifications do not prove that a participant is storing data, only that they have access to it. If the participant is accessing storage that is not yet being offered to Sia, it makes no difference to Sia where the storage is coming from. It is important, however, to prevent participants from storing data "recursively" - that is, storing their designated segments on Sia.

Participants must also be prevented from downloading requested segments from their own quorum.

Recursive storage is unprofitable for the participant, because they are getting paid to host 8 GB, but they must pay to store 8 GB with redundancy. If participants were also rewarded with newly minted coins, the situation might occur where the newly minted coins pay for the redundancy and leave profit for the dishonest participant. Therefore, newly minted coins cannot be distributed to participants who are already getting paid from another income source.

Preventing a participant from downloading segments from their own quorum requires a different tactic. To ensure high uptime, downloads cannot simply be restricted during the proof-of-storage process. Instead, this behavior is discouraged by making downloads more expensive than the reward for having the data. The actual cost of a download is unknown, and we do not want to manipulate the price, so the cost of a download is manipulated in a very specific way.

Each block, a participant uploads one segment. A segment is 64 KB, or 1/131,072th of the data that the participant is supposed to be storing. For successfully proving that they have the block, the participant will be rewarded for hosting 8 GB for the duration of one block. The cost of downloading the file for the proof needs to be equivalent or greater than the reward for storing the file. Depending on what the reward is, this cost can be calculated.

The choice of ring used in the storage proof may be known by a dishonest participant up to one block earlier than the honest participants. It follows that the premium cannot be added to just the ring that participants are required to prove. Instead, all downloads have this cost as a deductible. By the next block, honest participants will know which sector was required for storage proofs. Any sectors that were downloaded which were not used in proofs can be refunded their deductible. In rare cases, honest clients will download a sector that is being used for storage proofs, and will not be refunded. This is unavoidable, but is rare and is not even very expensive (on the order of fractions of a cent). Furthermore, it can happen a maximum of once per block per quorum. A client downloading many sectors at once will only be at risk of overpaying for at most one.

7 Delegation

Sectors are stored on Sia with redundancy. A dishonest set of participants can collaborate and store the file at a reduced redundancy while pretending to store the file at high redundancy. This problem is unavoidable as long as the network is allowed to be self repairing. A potential solution would be to encrypt each segment with a different algorithm, meaning that dishonest participants cannot reduce the redundancy. The downside is that honest participants cannot automatically repair the file if segments are lost.

An alternative solution being explored is the idea of false redundancy penalties. Hosts are supposed to be randomly distributed, meaning that even in the case of disasters like power outages or earthquakes, each quorum should lose at most one participant. If multiple participants disappear from a quorum at the same time, the network will treat them as falsely redundant. This would allow the network to levy heavy penalties and fines, but would not help a consumer to recover lost data. Other solutions are also being considered.

8 Random Number Generation

Sia's consensus algorithms require the use of random numbers. More importantly, each participant in a quorum must use the same set of random numbers. An attacker who can manipulate these numbers can launch a number of attacks against the network. It is therefore necessary to devise a means of securely generating a shared entropy pool. Random number generation is done in two stages. The first stage is the internal stage, during which participants engage in an algorithm that produces a nearly-random value. The second stage mixes in entropy from an external source to ensure randomness.

Each block, each participant generates a random string locally and submits it with their heartbeat. During block compilation, all of the random strings are concatenated and hashed again to produce a unique random seed. This is the internal step. The goal of the internal step is to produce a random seed

that is unique to each quorum. A single honest participant will make it computationally infeasible for a dishonest set of participants to control what seed is produced. Though dishonest hosts can manipulate their own entropy to avoid certain seeds, the result will still differ from the internal seed of any other quorum.

After the internal seed is determined, an external seed is acquired. Every quorum will be receiving the same external seed, but each quorum needs a unique set of random values, which is why the internal step is used. The internal seed is concatenated to the external seed and their union is hashed, creating a secure random seed that is unique to each quorum.

9 Quorum Integrity

A quorum is considered to have integrity if at least 20% of participants are honest. The network assumes that half of all participants are honest, and that participants are randomly distributed throughout the network. Given these assumptions, the probability that a given quorum lacks integrity is given by:

$$\sum_{i=102}^{128} \binom{128}{i} 0.5^i \, 0.5^{128-i} = 3.889 \times 10^{-12} \quad (1)$$

If only 45% of the network is dishonest, the probability is even more favorable:

$$\sum_{i=102}^{128} \binom{128}{i} 0.45^i \, 0.55^{128-i} = 9.415 \times 10^{-16} \quad (2)$$

Quorums that are more than half dishonest become vulnerable to an attack where honest participants can be evicted from a quorum for inactivity by not being in the block that is composed of the majority of the participants. Every time that a participant is evicted from the network for inactivity, it is assumed that dishonest participants have maliciously forced the eviction. Steps must then be taken to restore quorum integrity, lest all of the honest participants are evicted and only dishonest participants remain. For each participant kicked from the network for inactivity, two are forcibly removed from the quorum. Then all three get replaced with new participants. Assuming that only honest participants get dropped, kicking two random participants and replacing them with random participants will set the equilibrium to 2/3 dishonest participants. This is not a perfect solution, but it maintains integrity in the quorum.

If dishonest participants continually shove out honest participants, the quorum enters a state of turbulence. Presently there is nothing in the protocol to discourage dishonest participants from causing high levels of turbulence, though there are plans to make turbulence expensive to the whole quorum. We are not sure what the best solution is.

It should be noted that a quorum which is 2/3 dishonest is at risk of file loss. If all dishonest participants leave the quorum at the same time, they will be fined heavily. However, all files with a redundancy ¡3 will be lost from the quorum. This is also considered a weak point, and solutions are being considered that would enable a self repairing file to be spread out over multiple quorums.

A high-level, non-self-repairing solution exists for files that must endure extreme attacks, though it is not discussed in this paper. Because storage has legitimate utility and value, it is unlikely that an attacker would gain control of a full 50% of the network - there are too many parties that would be interested in contributing. Unlike Bitcoin, there are no mining pools, which means the risk of one pool gaining too much power does not exist. Nonetheless, we still attempt to make the protocol secure in the face of a full 50% attack.

10 Quorum Communication

Sia consists of many quorums each performing local consensus algorithms. Each sets the same price for storage, and when a new participant is added to the network, the quorums must collaborate to ensure it is placed randomly. For efficiency, quorums are organized into a tree structure. At the lowest level of the tree, each quorum has several siblings. Quorums only report aggregate values to their siblings. With regards to storage, only a few values matter:

- The number of participants in the quorum.
- The total balance of all sectors on the quorum.
- The total number of sectors in use on the quorum.
- All transactions attempting to rent storage from Sia.
- Any transactions announcing an intent to become a new participant.

Each time a quorum creates a new block, a meta-block is sent to all of its siblings. A meta-block contains the values above, along with a hash of the full internal block and a signature from each participant in the quorum. Meta-blocks are sent probabilistically, with each participant randomly contacting 3 participants in each sibling quorum. Assuming that each quorum has at least 20% honest participants, at least 75 messages will be sent, each having a 20% chance of being sent to an honest participant in the other quorum. This reduces the probability of failure to 1 in 100 million. It is also extremely unlikely that two quorums of only 20% honest participants will exist as siblings, increasing even further the chance of success.

It is possible that a dishonest participant or set of participants will try to announce a fake block. To prevent this, upon receiving a block from a sibling, a quorum will send out an confirmation by performing the same algorithm. With high probability, an honest host will see the dishonest block and be able to conjure a proof of the dishonesty. If a confirmation is sent out and no accusation of dishonesty is made within 1 cycle, the block is assumed to be valid.

Together, a set of siblings quorums form a meta-quorum. These quorums share aggregate resources in the same exact manner, forming a meta-meta-quorum. This layered quorum approach continues until there is only a single most-meta-quorum - the *root quorum*.

Meta-quorums need to maintain consensus, such that each quorum within the meta-quorum receives the same block. It is too expensive for quorums to engage in the same consensus algorithm that participants engage in, so an alternate solution with simpler constraints is needed. We can assume that at least 20% of all participant in each quorum is honest. Carefully executed, this gives us enough to trust any individual quorum. A random quorum is chosen each block to compile the block for the meta-quorum. This quorum will be a participant in the meta-quorum that it is compiling a block for, and will have the previous block as well as the current state of the meta-quorum. It will also, with very high probability, have all of the most recent blocks from the sibling quorums that form the meta-quorum. The honest participants in the quorum will be able to fill out the block and broadcast it to the meta-quorum. After a meta-quorum releases a block, it releases the meta-block for that block to all of its siblings. The exception is the root quorum, which has no siblings.

The broadcast algorithm will work probabilistically, and is still under construction. To prevent dishonest blocks from being accepted, there will also be a confirmation algorithm that operates probabilistically. The algorithm will be designed such that the block compiler for the root quorum will not have to handle more than a few messages per participant, confirmation messages included.

11 Joining Sia

When a host wants to join Sia as a participant or set of participants, it must first make a down payment. There are multiple behaviors that a participant can engage in, such as turbulence, that could be used as DDoS attack against the network. If any of these behaviors are detected, the participant is fined. The down payment is used to cover this cost. Assuming that there is no dishonest behavior, the host can reclaim the coins upon leaving the network gracefully.

The host makes the down payment on whatever quorum the host has a balance. The host can announce itself as any number of participants, so long as it can pay the full down payment for each. To maintain randomness on the network, each participant will replace a random existing participant on Sia. The participant that gets replaced will then either join a completely new quorum or fill in a gap where a participant has gone offline. If there are any quorums on Sia that are missing participants (due to

inactivity or dishonesty), the displaced participant is sent to fill the gap. If there are no known gaps, the displaced participant is placed in a queue. When there are 128 participants in the queue, a new quorum is formed. If there are more than 2 * 128 gaps in quorums on Sia, a quorum is sacrificed and its participants are used to fill the gaps.

12 Use Cases

Sia can be used as an efficient replacement to many of today's web services. One obvious example is BitTorrent. BitTorrent requires that every seeder host the entire file, resulting in unnecessarily large redundancy. Another issue is that torrents die once their last seeder leaves. Files on Sia are guaranteed to remain accessible as long as their corresponding sectors are paid for. Because anyone can pay for a sector, anyone can secure the longevity of a file on Sia without having to seeding it continuously. Sia is not free, but neither is BitTorrent. The cost of BitTorrent is seeding files, which consumes hard drive space and bandwidth. Anybody with both of these resources can participant in Sia as a host and gain siacoins, offsetting the cost of their downloads.

Sia also makes sense as the storage layer for any streaming application. By embedding images and videos into Sia instead of hosting them on a centralized server, providers can achieve lower storage costs, lower bandwidth costs, higher throughput, and equivalent latency. Sia's distributed nature makes it a pseudo-CDN, resulting in equal performance regardless of geographic location.

Sia is also elastic, meaning you only need to rent exactly as much storage as you are currently using. Instead of defensively buying a 1 TB drive, if you are only consuming 650 GB then you can rent exactly 650 GB. You can expand the volume of storage you are renting at any time, within minutes.

On sufficiently fast connections, Sia could potentially be used as a replacement for hard drives. The throughput on Sia can potentially exceed a gigabit per second, comparable to local storage. The bottleneck would be latency, which would still be around a hundred milliseconds per request. This problem could be minimized through local caching of frequently accessed files.

13 Remaining Features

The size of the network makes it infeasible for a quorum to store the locations of all other quorums. The current plan is to use a DHT, where quorums can look up the addresses of other quorums. The DHT needs to be designed to withstand high levels of dishonesty.

The financial system has not been fully explained. Intra-quorum transactions are easy, but sending transactions between meta-quorums has not been explained. The general idea is that a meta-quorum will tell its siblings about any transactions that cross borders. Transaction transversal needs to be carefully constructed such that very little communication happens through the root quorum.

An external source of entropy is needed. One possibility being explored is a new cryptocurrency specifically designed to produce entropy. This may be the subject of a future paper.

Participants within a quorum need synchronized clocks. We do not yet have an algorithm to achieve this, though there is a Leslie Lamport paper that solves a very similar problem.

It's not explained where new quorums are added to, and where sacrificial quorums are sacrificed from. Most likely, for security reasons, they will be added to and sacrificed from random places. The meta-quorum tree will need to be balanced, however.

Wallets are expected to have a scripting system built in. This scripting system has had very little thought put into it thus far, because it's more of an "icing on the cake" than a vital part of Sia.

The algorithms presented have not been under much scrutiny. We need experts to inspect the algorithms and proofs presented in this paper to verify that there are no errors in the logic or attack vectors that have not been accounted for.

Appendix: Vocabulary

Segment
An erasure coded piece of a physical file held by a single participant.

Sector
A logical block of data stored on a quorum. This logical block is composed of a set of equal sized erasure coded segments, one segment held by each participant in the quorum housing the Sector.

Ring
A physical block of data; the product of erasure coding a sector. Rings are distributed across a quorum.

Participant
A network-connected computer offering a discreet and inflexible volume of storage to the network. A participant is the local representation, and contains data such as private keys.

Sibling
A public representation of a participant. Private data such as private keys is not contained in a sibling.

Quorum
A set of randomly chosen siblings working in consensus to monitor a subset of the Sia network.

State
The status of a quorum-monitored subset of the Sia network. Every participant in the same quorum will have an identical state. Every quorum has a different state, as every quorum monitors a different subset of the Sia network.

Meta-Quorum
A set of quorums or meta-quorums acting in consensus.

Root Quorum
The highest level quorum. The root quorum makes all of the major network decisions, such as file price.

Block
A set of updates to the State of a particular quorum.

Heartbeat
A set of updates from a single participant in a quorum. A block is actually just a list of heartbeats, one from every participant in the quorum that the block acts upon.