Sia : Decentralized, Compensated, Self-Repairing Computer Storage

David Vorick
Nebulous Labs

Luke Champine
Nebulous Labs

May 6, 2014

Abstract

Sia is a decentralized platform for computer storage acting as a marketplace for storage. On the supply side, host computers contribute storage in return for Sia's internal cryptocurrency - Siacoin. On the demand side, consumers use Siacoin to rent storage space. The price of storage is set by a market making algorithm that is sensitive to the volume of supply and demand. Files on Sia are distributed to a large number of randomly selected hosts. Erasure Coding is used to give the files redundancy, enabling files to be spread over hundreds of machines with highly reliable network redundancies as low as 1.2. Spreading the files over hundreds of hosts means that files can be downloaded in parallel from hundreds of places and achieve high throughput. As explored in the paper, Sia should be a cheaper and superior solution to cloud storage when compared against the existing centralized alternatives.

1 Introduction

Sia is a decentralized platform for storing data on the cloud. The storage on Sia is block level, like an unformatted hard drive. This makes Sia file system agnostic - you store information on Sia by setting and requesting arrays of bits. It is very easy to put a file system such as NTFS or ext4 on top of Sia. It is expected that the average user will not interact directly with Sia, but rather with an application such as a music player that streams directly from Sia.

Because Sia operates at the block level, it does not make as much sense to talk about files being stored on Sia. Instead we talk about sectors of storage, a sector being a logical set of bits stored on Sia. Files can be stored within sectors, across multiple sectors, or any other way that a file can be translated into a series of bits. Sia is random access, which means you can download any part of a sector without needed to download the whole sector.

Each sector is given to 128 machines that monitor the health of the sector. If any of these machines goes offline, it is replaced by the network, keeping the sector available and in good health. The sector is broken into 128 segments, of which M are redundant, where M is chosen by the uploader. Any M machines can go offline at the same time and the sector will still be recoverable - this is because Reed-Solomon coding is used. If a machine goes offline or fails, it will be replaced by a new machine within a few hours, bringing the sector back up to its original redundancy. Having a sector split between 128 machines and being able to tolerate M faults every day means that even in scenarios of high paranoia M can be relatively small.

Reed-Solomon codes are maximum-distance separable. This means that redundancy is maximally efficient. For example, take a 100mb sector that gets broken up into 128 pieces with M=28. You have 100 non-redundant pieces and 28 redundant pieces. Of these, ANY 28 pieces can be lost and the full sector can still be recovered. Together, all 128 pieces consume 128mb of physical storage. The default value for M on Sia is 28.

Each sector is stored on 128 machines, and the sector can be downloaded in parallel from all of them. This allows Sia to obtain very high throughput for downloads, even if each machine is only offering a lit-

tle bandwidth. Additionally, for situations requiring reduced latency, only the closest 128 - M machines need to be contacted. Because a random set of machines are hosting each sector, the closest 128 - M machines are likely to be very close when M is large, regardless of the location of the downloader.

Each sector stored on Sia is accompanied by approximately 40kb of overhead that allows the network to monitor and error-check the sector. Multiple files can be stored on a single sector; 40kb is not a minimum file size it is a minimum sector size. Each sector is broken into 128 segments after redundancy is added. The maximum size of a segment is limited to 1mb. The maximum size of a sector therefore depends on the value of M, a larger M means a smaller maximum sector size. Sectors are just a way for the network to orgainze, even though the maximum sector size is small, files and information can be stored across many sectors. Optimizing for smaller sectors means that each file is stored on that many more machines, which means the benefits of parallel downloads are increased.

Another property of Sia is that it becomes very difficult to censor data. Forcefully taking sectors off of the network requires corrupting or destroying more than M of the 128 segments that compose the sector. Because sectors on Sia are randomly distributed among a global set of hosts, it is very difficult for a single entity (even as powerful as the NSA) to attack a sufficient number of machines to corrupt a segment, especially when M is set to be large.

This network design results in some great features:

- High Speed
- Low Latency
- High Robustness and Security
- Low Overhead

2 Managing Sectors

When uploading, a sector can either be encrypted or public. Encrypted sectors are great for housing personal or private files. Public sectors are great for hosting embeddable web content such as images and videos. Any sector can be downloaded by anybody, and a sector may end up on any malicious machine. The best way to protect a file or sector is to use client-side encryption.

Sectors can also be mutable or immutable. The contents of a mutable sector can be altered at any time, but only by authorized parties. Authorization is done using public key cryptography. It is expected that most sectors will be mutable. Immutable sectors cannot be altered after they have been uploaded. This is achieved by authorizing nobody to edit a file. Immutable sectors are appropriate for public and/or shared data - if you use a public sector, you want assurance that the data will be the same every time you access it.

Each sector is associated with a balance of siacoins, which is used to pay for the storage that the sector consumes. Each block, a volume of siacoins is subtracted from the sector balance according to the current price of storage. When the balance runs out, the sector is deleted, and space on the network is cleared for new data. Anybody can add to the balance of a sector at any time, whether the sector is mutable or immutable. Immutable sectors cannot be deleted from the network unless the siacoin balance fully depletes.

3 Use Cases

Sia can be used as an efficient replacement to many of today's web services. One obvious example is BitTorrent. With BitTorrent, every seeder needs essentially the entire file that's being seeded, resulting in a huge global redundancy of the file. With Sia, a file only needs a small amount of redundancy to provide the same amount of throughput. Another issue is that torrents die once the last seeder disappears. The half life of a torrent can be as low as a few months. With Sia, a file will stay around as long as the sectors are paid for. Because anyone can pay for a sector, anyone can secure the longevity of a file on Sia without needing to go through the hassle of 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 and gain Siacoins, enabling them to also download from Sia. Sia is a comprehensive replacement for BitTorrent.

Sia makes sense as the storage layer for any streaming application. This includes content embedded into web pages. By embedding images and videos into Sia instead of hosting them on a centralized server, you can achieve lower storage costs, lower bandwidth costs, higher throughput, and equivalent (or even reduced) latencies. Sia automatically acts as a CDN, meaning users across the world receive similar service to local users. Sia could also be used for non-streaming applications, such as for save data in a video game, or even for loading new areas in a video game.

Sia is also elastic, meaning you only need to rent exactly as much storage as you are currently using. Instead of needing to plan for the future and buying a 1TB drive, if you are only consuming 650GB then you can rent exactly 650GB of Sia storage. 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, which is nearly as much throughput as an SSD. The bottleneck would be latency, which would still be around a hundred milliseconds per request (except for sectors with very high redundancy). By moving enough of your operating system into RAM, you can avoid this problem and replace your local hard drive.

4 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 emphasize the current hosts on the network as opposed to the early adopters. Initially, the number of coins generated will be very high, and the early adopters will be rewarded. In the long run however we want the people actively contributing resources to the network to be the people that are most heavily rewarded. We also believe that an inflationary model will result in a currency that is much less volatile.

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. All income given to hosts has a fee, a host income fee of 3.9%.

The developers look to feed themselves and reward the early backers of Sia by using a new tool called Siastock. Siastock is a 100% premined currency that will be initially owned completely by the developers. During an IPO or a pre-IPO, some percentage of Siastock will be traded for seed money. Someone who owns 1% of all Siatock will receive (1% * 3.9%) = 0.039% of all host income on the Sia network. This model of developer and backer revenue has been chosen instead of premining because it enables us to make Sia inflationary while still having a guaranteed income. It also makes it clearer exactly how much value is going towards the developers and stock owners.

The only way to access the storage on Sia is to use siacoins. The storage on Sia has a certain value, and the only way to access that value is to use Siacoin. 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 internal value of Sia, combined with the fact that this value can only be accessed using Siacoin gives Siacoin an internal value that is roughly equivalent to the value of all the resources on Sia. Siastock provides a guaranteed income that is equivalent a percentage of all host income in siacoins, Siastock also has an internal value.

5 Quorums

When a machine joins the network, it joins the network as a set of participants. Each participant contributes exactly 8gb to the network. Each participant is then assigned to a random quorum on the network. A quorum is a set of 128 participants acting in consensus. Each quorum has a 'State', which is the representation of the current state of the quorum. For example, the 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 has one update, called a 'heartbeat', from every participant. Each block, every participant is required to submit a heartbeat containing a minimum amount of information. This information 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', which is a heartbeat followed by a list of signatures of 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 send 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. During step 1 for example, a heartbeat only needs to be signed by the participant producing the heartbeat. If during step 1 a heartbeat is received with 2 or more signatures, this is acceptable, we only care that it has at least 1 signature. There are 128 steps total. If a participant signs two different heartbeats from itself, that participant is dishonest and will be thrown from the network. Because participants releasing more than two heartbeats are treated the same as participants releasing exactly two heartbeats, honest participants do not need to share more than two heartbeats per participant. 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. For each signed heartbeat that a participant receives
   (a) If the number of signatures on the signed heartbeat is less than the step counter, ignore.
   (b) If there are multiple signatures from the same participant, ignore.
   (c) If there are signatures from non-participants, ignore.
   (d) If it is a heartbeat that has been seen before, ignore.
   (e) If we have already seen two other heartbeats from the creator of this heartbeat, ignore.
   (f) Add signature to the signed heartbeat and send it to all other participants in the quorum.
4. When the step counter reaches 128, take all of the heartbeats and compile them into a block.

The concern of the honest participant is getting the exact same set of heartbeats as every other honest participant in the quorum. If all honest participants follow this algorithm, then the premise will hold. 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 in the quorum. There is one exception if 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. This is okay however, because the action taken by an honest participant will be to throw the dishonest participant from the network, regardless of the contents or quantity of the heartbeats.

6 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. Then the proof-of-storage algorithm is run. This extracts the data for the proof-of-storage 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 also parsed and verified as honest. Any participant that fails the storage proofs and/or random number proofs are thrown from the network and their heartbeats are discarded.

All that remains to be parsed is transactions. Before processing the transactions, the honest heartbeats are sorted into a random order. The shuffle algorithm (explained more fully in section 'Random Number Generation') uses a deterministic seed from the previous block, such that all participants will arrive at the same random ordering. Then, heartbeats are processed one at a time, taking the transactions from within the heartbeat and processing them 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 the heartbeats in the same order. Therefore, all participants will end with the same new state.

7 Proof-of-Storage

Each participant is responsible for storing 8gb. This 8gb is divided into many segments, each of maximum length 1mb. 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 as proof-of-storage must upload their entire segment, which can then be hashed and verified. This means that each block, up to 128 segments of size 1mb might be uploaded in the block. This represents a fair amount of bandwidth, but blocks are not kept as a part of Sia, so it does not create a bloated blockchain.

If a participant is not storing anything (or is only storing the hashes), the proof-of-storage algorithm will detect their dishonest every time. If a participant is storing only half of the data, then each block the proof-of-storage algorithm has a fifty percent 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 something. 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 approxi-

mately 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 it's 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 done 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 don't actually prove that the participant is storing data. Instead, they prove that the participant has access to the data. 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. We do however want to prevent a participant from claiming to bring new storage to Sia, but then instead store the files they are supposed to have in another place on Sia. We also need to prevent a participant from downloading the segment they need to prove from their own quorum.

First we will address recursive storage, where a participant claims to store data but instead rents out space on another quorum. This is unprofitable for the participant, because they are getting paid to host 8gb, but they must pay to store 8gb 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. We want to avoid this, so newly minted coins cannot be distributed to participants who are already getting paid from another income source.

Preventing a participant from downloading the whole file from their own quorum requires a different tactic. The information necessary to recover data is already available in the local quorum, it only needs to be downloaded. Furthermore, Sia has a policy of being always available, which means a dishonest participant can download a sector at any time and figure out what they need. This behavior is prevented 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 64kb, 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 8gb 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.

Which ring will be uploaded for proving storage may be known by the dishonest participant before it is known by the honest participants (up to one full block early). So 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.

(Please give me feedback for this section, I realize it's complicated and difficult to read.)

8 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.

One 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.

9 Random Number Generation

Random numbers are important to Sia, an attacker who can manipulate the random numbers produced in quorums and throughout Sia can launch other attacks against Sia as well. Random number generation is done in two stages. The first stage is the internal stage, where participants engage in an algorithm that produces a nearly-random value. The second stage grabs entropy from an explicit external source - Aiza - to solidify 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 they can manipulate their own entropy to avoid certain seeds, the only goal is to make sure that the internal seed is not the same as the internal seed of any other quorum.

After the internal seed is determined, an external seed is acquired from Aiza, a cryptocurrency designed to generate secure public random numbers. 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. After getting entropy from Aiza, the internal seed is concatenated to the external seed and their union is hashed. This will create a secure random seed that is unique to each quorum.

10 Quorum Integrity

A quorum is considered to have integrity if at least 20% of participants are honest. Sia as a network assumes that half of all participants are honest. Sia also assumes that participants are randomly distributed throughout the network. Given this, there is a 3-in-one-trillion chance that a given quorum will not have integrity, as demonstrated by this equation:

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

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

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

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 great, 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 implement something that makes 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 less than 3 full redundancy 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, I find it unlikely that an attacker would get up to a full 1/2 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.

11 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. For example, a quorum will report how much storage is being rented, how many participants are active, and the total balance of all the wallets in the quorum.

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 above 5 bullet points, 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, reducing the chance of failure to one in 100 million. It is also extremely unlikely that two quorums of only 20% honest particpants 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. To simplify the vernacular, the smallest quorum is called 1Q. A meta-quorum is called 2Q, meta-meta 3Q, and so on. The largest quorum on the network is called RQ, or the "Root Quorum". The general term for larger quorums is meta-quorums.

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 tha 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. This, with careful execution, gives us enough to trust any individual quorum. A random 1Q quorum is chosen each block to compile the block for the meta-quorum. This 1Q 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 dishonst blocks from being accepted, there will also be a confirmation algorithm that operates probabilistically. The algorithm will be designed such that the 1Q block compiler for the root quorum will not have to handle more than a few messages per participant, confirmation messages included.

12 Joining Sia

When a machine 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 DDOS Sia. If any of these behaviors are detected, the participant is fined. The participant must have a balance of coins that can be fined. This is why machines must make down payments before joining Sia. Assuming that there is no dishonest behavior, the machine can reclaim the coins upon leaving the network gracefully.

The machine makes the down payment on whatever quorum the machine has a balance. The machine 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 (say 500 dishonest participants join the network at the same time), 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 put into 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.

13 Remaining Features

A quorum cannot store the locations of all other quorums in the network, that's too much information and too much to keep track of. The current plan is to have a DHT where quorums can look up the addresses of other quorums. The DHT needs to be build with the assumption that there is a large volume of dishonesty.

The financial system has not been fully explained. Intra-1Q transactions are easy, however 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 communcation happens through the root quorum.

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 sacrifical 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 scriting 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 erasure coded data. A ring is every segment for a particular file in a quorum. A ring is what the data looks like after it has been erasure coded; a sector is what the data looks like before it has been erasure coded.

Participant
A network-connected computer offering a discreet and nonflexible volume of storage to the network.

Quorum
A set of randomly chosen particpants working in consenses to monitor a fragment of the Sia network.

State
The status of a quorum-monitored fragment 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 fragment of the Sia network.

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

Root Quorum
The highest level quorum that 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.