Truncating Blockchains with Tangly Statistics

Nathan Stouffer advised by Dr. Mike Wittie

Abstract

Blockchains have applications in cryptocurrencies, supply chain tracking, and providing data integrity. A blockchain provides an immutable, decentralized record. To provide immutability, some blockchains use Proof of Work to construct an ever growing chain of blocks in a peer-to-peer network. Participants, known as miners, expend computational resources to create new blocks. By design, users trust the longest blockchain and blocks are intentionally difficult to create. As a result, data far back in the blockchain is immutable.

By design, miners have collective, but not individual, control over a blockchain. When miners are sufficiently decentralized, it is infeasible for a coalition of miners to gain explicit control of a blockchain. Explicit control would allow a coalition to decide which blocks are added to the chain, reducing the integrity of the system. Thus a blockchain performs better when control of the chain is decentralized. Decentralization can be difficult to achieve when blockchains grow to an unmanageable length. To begin mining, one must download and process the entire chain (a process called bootstrapping). This is not a problem when the chain is relatively small, but a larger chain (such as Bitcoin) can take days to download and process. Such a long chain prevents lightweight devices from becoming miners and incredibly long bootstrapping time deters participation from many users who do have sufficient space. Together, these issues make joining a long blockchain more difficult.

Over the summer of 2020, I worked with collaborators to devise a faster bootstrapping method. At a high level, our solution has miners of recent blocks vote for a blockchain summary. If sufficiently many votes agree, bootstrapping nodes can trust the blockchain summary and no longer need to download the entire chain. This drastically reduces bootstrapping time. For my capstone project, I continued the solution from the previous summer and implemented a model of the solution. Using the model, I ran simulations and extracted experimental results.

1 Introduction

Blockchain was introduced as a way to provide distributed consensus without relying on a central party [1]. Bitcoin was released to the general public in 2009 as the first implementation of a blockchain. Bitcoin quickly garnered a lot of support and became one of the most prominent cryptocurrencies. Since Bitcoin's inception, blockchains have found many applications. They are used in other cryptocurrencies, verify supply chains, and increase device autonomy in the Internet of Things [2]. Blockchains are even being used to provide security and privacy in the medical field [3]. Blockchain technology has the potential to enhance data security in a wide range of diverse applications.

Blockchains are decentralized. Avoiding a central party provides two benefits. First, there is no central location for an attacker to compromise, giving blockchain a robustness that centralized systems struggle to achieve. This is a nice property because no centralized attack (such as attempting to flood a server with requests) can succeed against a blockchain; it is just too large. Second, there is no single party with complete control over a system, preventing a monopoly of power. An example of where lacking a central party is useful is international currency exchange. Banks have the power to charge high transaction fees for international exchanges and customers are forced to pay the fees because there is no alternative. If one transfers money over a cryptocurrency, there is too much competition between parties (miners) for a monopoly to emerge. Since there is no monopoly, the market finds a fair equilibrium between miners and users.

Blockchain's services are best realized when control of the chain is decentralized. Decentralized control can become difficult to achieve when blockchains grow to an unmanageable size. Joining a well-established blockchain can take days, excluding lightweight devices and reducing the likelihood a large device joins. This means a blockchain is less decentralized and prevents Internet of Things devices from participating in a blockchain network. This project provides and analyzes a new, off-chain, efficient, and secure method for joining a blockchain. This project contributes towards making blockchains more effective and increasing their decentralization. More effective blockchains can extend data reliability and user privacy [2] [3].

The remaining sections provide background information, our solution, methodology, results, and a discussion.

Nathan Stouffer

email: nathanstouffer1999@gmail.com
phone: 208.293.5883
github: github.com/nathanstouffer

A motivated student graduating in Spring 2021 with a double major in computer science and mathematics. Hoping to join a fast-paced and exciting work environment where I can combine both my majors to develop software.

EDUCATION

Computer Science Major and Data Science Minor — *Bachelor of Science*

GRADUATING SPRING 2021

Mathematics Major — *Bachelor of Science*

GPA: 3.95

Montana State University (MSU) – Bozeman, MT

Relevant coursework: advanced algorithms, networks, software engineering, computer security, computer graphics, machine learning, computational geometry computer science theory, abstract algebra, topology, dynamical systems

EMPLOYMENT

Math Tutor — *MSU*

FALL 2018 – PRESENT

Working as a tutor in the Math Learning Center at MSU
Identifying difficult areas for students and explaining problem solving techniques
Assisting students with algebra, pre-calculus, calculus I/II, and linear algebra

Research Experiences for Undergraduates — *MSU*

SUMMER 2020

Participated in a Research Experiences for Undergraduates (REU) program
Communicated with collaborators about complex technical problems and ideas
Worked towards truncating a blockchain network's ever-growing chain

Computer Science Course Assistant — *MSU*

FALL 2019

Instructed a lab section of 24 students in an introductory computer science course
Encouraging students to think through problems and own their solution
Hosted weekly office hours
Graded lab assignments and larger programs

PROJECTS

Senior Capstone — *MSU*

FALL 2020 – PRESENT

Implementing and extending the research performed in my REU program
Working with collaborators to produce a new protocol that prunes blockchains
Building and interpreting results from a large-scale model of our solution

Emergent Behavior — *MSU*

SUMMER 2020 – PRESENT

Exploring the emergent behavior of agents acting on local rules
Building an agent-based computer simulation
Analyzing bifurcations in emergent behavior based on initial conditions
Comparing results with a partial differential equation model

Directed Reading Program — *MSU*

SPRING 2020

Independently studied a book about abstract algebra
Discussed thoughts and questions with two graduate students once per week

SKILLS

Java, Python, C++, C, OpenGL, MATLAB, LaTeX, Git

ACCOMPLISHMENTS

MSU Math Department's Outstanding Scholar Award

MSU's Achievement Scholarship

Milton F. Chauner Mathematics Excellence Scholarship

Mary C. Griffin Scholarship

President's List (MSU): S18, F18, F19, S20, F20

Dean's List (MSU): F17, S19

1st place team at MSU's 2019 programming contest

2nd place team at MSU's 2018 programming contest

Ran a marathon

Voted "Most Influential (2018–2019)" by MSU's National Residence Hall Honorary

ADDITIONAL EMPLOYMENT

Resident Assistant — MSU
Construction Laborer — ID

INTERESTS

skiing (downhill/xc)
ultimate frisbee
running
cribbage

2 Background

2.1 Blockchain

At a high level, a blockchain is a sequence of blocks that is immutable in practical applications. Immutability in blockchains is often provided through Proof of Work. Proof of Work was introduced in [4] and first used for blockchain in the Bitcoin whitepaper [1]. In a Proof of Work blockchain, a block is valid if its cryptographic hash is below a preset threshold, determined by a blockchain's community. It is typically set such that mining a block takes the same amount of time, on average, throughout the life of a blockchain. The threshold will adapt as blocks are created to meet this requirement. Each block primarily consists of a payload of data that is recorded (financial transactions, medical records, etc). In addition to the payload, a block records the hash of the previous block, tying the blocks together in a chain. Every block also contains a field for a fixed-length string of arbitrary bits. The fixed length string is called a nonce [1]. "Mining" a block consists of searching for a nonce that results in the hash of the block being below the threshold. By design, it is difficult to find such a nonce because the hash function is cryptographic. A miner's best strategy is to guess and check nonces until finding a sufficiently low output. In cryptocurrencies, miners are awarded currency for their work. Figure 1 displays the typical structure of a blockchain.

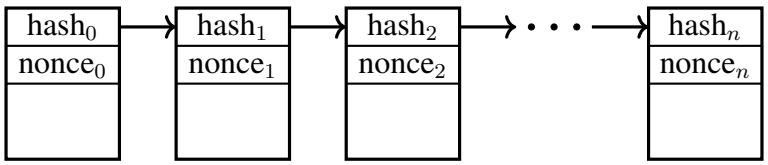

Figure 1: Prototypical Proof of Work Blockchain

A broader perspective of a blockchain is an implementation of a finite state machine. In the state machine, blocks are transitions between the states of applications running on top of a blockchain. Abstractly, the k^{th} block B_k is a transition from state S_{k-1} to state S_k with certain validity requirements. See Figure 2 for a visual.

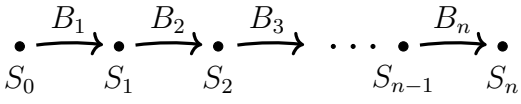

Figure 2: Blocks as transitions between states

A tangible example is a cryptocurrency. Cryptocurrencies realize state as a record of the amount of currency each public key controls. Blocks are a set of transactions that transfer currency between public keys (analogous to accounts). In state S_0, no one controls any currency. Then block B_1 is

created and its miner is granted some currency. This moves the blockchain to S_1 which reports how much currency that miner controls. Then block B_2 is mined, granting cryptocurrency to its miner. Additionally, block B_2 could contain a currency transfer from the miner of the first block to another user. Then the state S_2 records how much funds each user of the cryptocurreny controls. Then block B_3 is created and the sate machine is updated with the changes in "account" balances. This process repeats until state S_n is reached, which contains the current account-balance pairs of all the current users of a cryptocurrency.

2.2 Technical Problem

To understand the technical problem, we must understand two properties of blockchains. First, miners must obtain every block before mining new blocks. And, second, blockchains grow in perpetuity. These properties mean that new miners must download and process an ever increasing amount of data as time passes. Even now, when Bitcoin is just over a decade old, it can take days to become a miner. Such a barrier can deter large machines from the joining the chain and entirely prevent small machines from mining.

The root of the problem is that mining requires all the information in a blockchain. Our goal is to produce an off-chain, efficient, and secure method to bootstrap miners in a blockchain. An off-chain solution is one that does affect the underlying blockchain. This is important because many on-chain solutions to this problem cause a hard fork in a blockchain. A hard fork is when a blockchain splits because miners disagree on the format of a block [5]. While a blockchain can survive a hard fork, the mining power splits and weakens the blockchain. We want an efficient solution so that our idea could be useful in an actual blockchain. Finally, security is important because blockchains often record important information such as medical records. Our solution should not introduce vulnerabilities to such important information.

To frame this project, we use the Goal Question Metric approach presented in [6]. This results in "the specification of a measurement system targeting a particular set of issues and a set of rules for the interpretation of the measurement data [6]." The model has three levels. At the conceptual level, a goal is defined. A goal consists of a purpose, an issue, a process, and a viewpoint. At the operational level, a set of questions are asked that characterizes the goal. At the quantitative level, data is associated with questions in an attempt to answer them. The data can be either objective or subjective. Table 2 displays the Goal Question Metric Approach we will use for this project.

Goal	Purpose	Decrease
	Issue	required time and space
	Process	bootstrap a node to a blockchain network
	Viewpoint	the bootstrapping node
Question	Q1	Is the solution off-chain?
Metrics	M1	Number of changes to blockchain protocol (must be 0)
Question	Q2	Is the solution efficient?
Metrics	M2	Asymptotic analysis of time and space requirements as the chain grows
	M3	Bytes of network traffic generated
Question	Q3	Is the solution secure?
Metrics	M4	Theoretical probability that a malicious actor fools a bootstrapping node
	M5	Empirical probability that a malicious actor fools a bootstrapping node

Table 1: The Goal Question Metric approach for this project

2.3 Related Work

Truncating the required bootstrapping information in a blockchain is an active area of research. The remainder of Section 2 is devoted to summarizing existing ideas and solutions.

The Bitcoin whitepaper [1] introduces the concept of lightweight nodes. Instead of becoming a full node, lightweight nodes only store the header chain and query full nodes to see if transactions are valid. This is a step in the right direction because the header chain downloads quickly, but it does not grant nodes the ability to mine new blocks. Nodes can only verify old transactions.

Another idea is to periodically place a summary block in the original blockchain [7] [8]. A summary block is responsible for reporting the net change caused by a certain amount of previous blocks. This process can be made recursive to any depth by creating a summary block responsible for a set of summary blocks (a summary of summaries) [7] [8]. Trust in the summary blocks is built in the same fashion as trust in the regular blockchain. A node joining the network would download the header chain and relevant summary blocks. If a summary block has been accounted for by another summary block, there is no need to download it. This idea is nice because it relies on the same security principles as blockchain, but the implementation would require a hard fork.

A different idea would be to use mini-blockchain [9]. A mini-blockchain requires that a block B_k includes the hash of S_k, cryptographically tying the state to the blockchain. A bootstrapping node can then request the header chain, a recent state, and blocks following the recent state. The new node can verify that the received state corresponds with the blockchain and then compute the current state with the information in the newest blocks. While this solution is quite clean, it would cause a hard fork.

Another way to provide state would be to create a new blockchain that records state [10]. The

new blockchain records the state of the original chain every so often and bootstrapping nodes can eliminate the need for blocks prior to the most recent summary. Since each state exists independently of all other states, the size of the second blockchain will not bloat like the original chain [10]. While this solution does operate off-chain, it is not necessarily secure. It is unlikely that a miner will give up valuable computing power to contribute to the second blockchain. This could allow a powerful, malicious actor to sabotage the second blockchain, making it unreliable.

The final related idea holds an election to verify state [11]. Miners are allowed to vote for a state and then bootstrapping nodes trust the majority of votes. Each vote is recorded in a block and each block has the capacity to store a single vote [11]. Storing votes in blocks makes them immutable, so a bootstrapping node is able to collect every vote. The issues with this idea are two-fold. First, a vote can only be generated as quickly as new blocks. In Bitcoin, this averages to about 10 minutes per block [1], meaning a bootstrapping node might have to wait a week for there to be sufficiently many votes to trust a state. Second, [11] relies on implementation details in the Bitcoin protocol and do not apply to blockchains in general.

3 Solution

3.1 Idea

Recall that our goal is to devise an off-chain, efficient, and secure bootstrapping method. The primary barrier to this goal is that blockchains require that a miner to obtain every block in the chain. Viewing blockchain as a finite state machine allows us to reduce the amount of information that a bootstrapping miner must obtain. To compute the current state S_n, one need only know is some state S_k and every block B_j with $j > k$. In standard blockchains, S_k is S_0 and the bootstrapping node needs every block in the chain. If S_k is close to the end of the chain, then the miner only needs the blocks following S_k, which will download much faster!

At this point we have reduced the problem of bootstrapping to providing some recent state S_k of the blockchain. How does a bootstrapping miner obtain the state S_k? They can't just ask a blockchain node, because the node might lie and give a false state. To solve this, we draw from [11] and host an election at regular intervals (say every 1000 blocks). We allow recent miners to vote. Note that miners are identified by the public keys in the blocks they mined, preventing identity theft. If the election is for state S_k, then a miner computes the hash of their version of S_k. The miner submits a digital signature of the hash; the signature should use the private key from the block the miner created. The digital signature prevents a malicious actor from forging the vote.

The election is decided based on the relative vote distribution. Let V denote the total number of voters (not the number of votes cast). For a tuneable $\beta \in (0.5, 1]$, we will decide an election if greater than $\lceil \beta V \rceil$ votes agree. However, we do not have every vote since voting is not required and

votes can be deleted by malicious actors. Instead, we have a sample of votes. How can we figure out whether the sample was drawn from the required distribution? The χ^2 goodness of fit test is a way to measure the probability that a sample comes from an expected distribution. If the probability returned from the χ^2 test is above a tuneable threshold α, we accept the state with the most votes.

The votes are submitted to a Distributed Hash Table (DHT). Storing votes in a DHT is a compromise: it is convenient because miners can vote at their leisure, but a powerful, malicious actor can delete votes. To combat vote deletion, we draw on the idea of a tangle [12]. A tangle is a directed acyclic graph where the vertices are added to tangle by selecting the two most recently added vertices as parents. In this context, vertices are realized as votes.

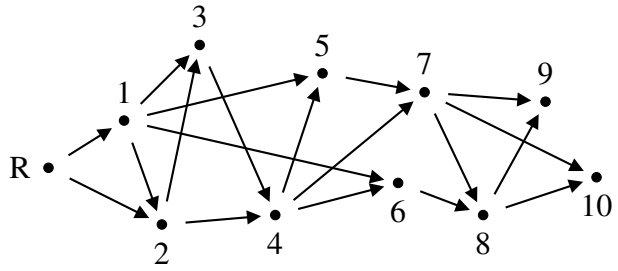

Figure 3: Example of a tangle

A set of DHT nodes are designated as tangle managers, each given complete control of their own tangle. A voter is expected to submit their vote to a deterministic subset of the tangles. A tangle manager receives votes and inserts each of them to their tangle by attaching them to parent votes. If a vote is not in every tangle of the deterministic subset, then it is considered invalid. The children of an invalid vote are also invalid. Then, deleting a vote causes a chain reaction that invalidates all the descendant of the deleted vote. The distribution of states that the descendants support matches the distribution of cast votes, so a malicious actor cannot swing an election by selectively deleting votes. A bootstrapping node can request the vote tangles and then make a decision based on valid votes.

3.2 Implementation

This capstone project will include a model implementation of the solution. The implementation will run a full-fledged DHT running on top of a network simulator. Scripted miners will submit votes to the DHT and bootstrapping nodes will request the tangles and decide the election. Using the simulation, we can gather empirical results indicating the security of our solution. The network simulator will allow us to gather information about the amount of network traffic required to facilitate an election.

4 Methodology

Much of this project is devising a method to bootstrap blockchain nodes. Since this is a conceptual task, it cannot be replicated. However, the experimental verification of our protocol can be repli-

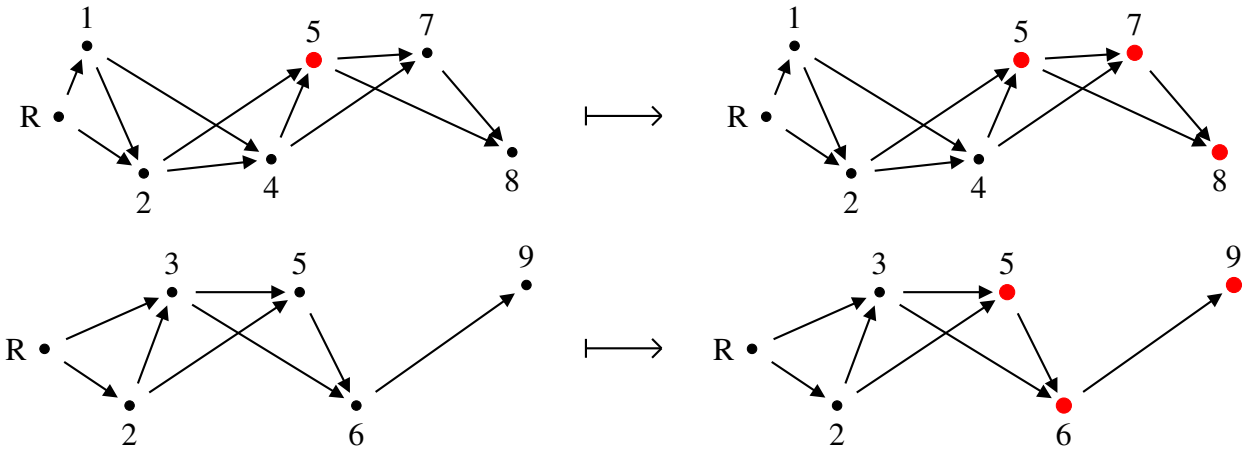

Figure 4: Showing a single invalid vote poisoning a set of tangles

cated. Section 4 is devoted to providing the necessary information for someone to replicate our implementation.

In order to match our model as closely as possible with our implementation, we chose to implement our solution on a network simulator using a full-fledged DHT. This is not strictly necessary but we felt that it made our results more applicable to a real implementation of our idea.

A neglected detail of the solution is the process of selecting parents in the tangle. The tangle manager has complete control over *which* votes are selected as parents, but we require that the parents are digitally signed in the vote. This means that the manager must tell the voter which votes are the parents and the voter includes that information in their vote. While this is some overhead communication for our solution, it prevents a malicious actor from changing the structure of a tangle to their advantage.

The most important aspect of the implementation is the intelligence of the attackers. We expect the constraints of our solution to prevent an attacker from *successfully* fooling a bootstrapping node. But the only way we can verify this is to simulate intelligent attackers. We have divided attacks into three categories: creating votes, deleting votes, and reordering votes. An accurate replication of this project should at least implement these strategies. Implementing more strategies is welcome, since it provides more information on our system!

We used the χ^2 test as follows. Consider some sample of votes S and our expected proportion of agreeing votes β. Our distribution comes in the form $(majority, other)$ where $majority$ references the number of valid votes that supported the most popular state and $other$ is the number of valid votes that did not support the most popular state. We have an expected distribution $(\beta|S|, (1-\beta)|S|)$ and an actual distribution of (S_{maj}, S_{oth}). Then we can use the χ^2 test to determine the probability that our sample comes from the expected distribution.

4.1 Design

The design pattern that appears in this project is the Strategy Pattern. The design pattern presents itself in the sorting styles used to generate the charts in Section 5. We have no canonical ordering for the votes in a tangle so we implemented a number of them and chose between them. To make the code cleaner, we use the Strategy Pattern. We have an IVoteSorter interface with a method called *sortedVotes()* that returns a sorted array of vote keys. Each class implements a different partial ordering that allows us to to sort the votes. The class RndSorter implements sorting by the vote's round. BFSSorter returns the order votes are encountered in a breath first search. Finally, DescSorter returns the votes in the order of the number of descendants each vote has. For the final charts, we decided to use the RndSorter class but the design pattern helped keep our code clean and organized. Figure 5 shows a visual of our Unified Modeling Language diagram for this pattern.

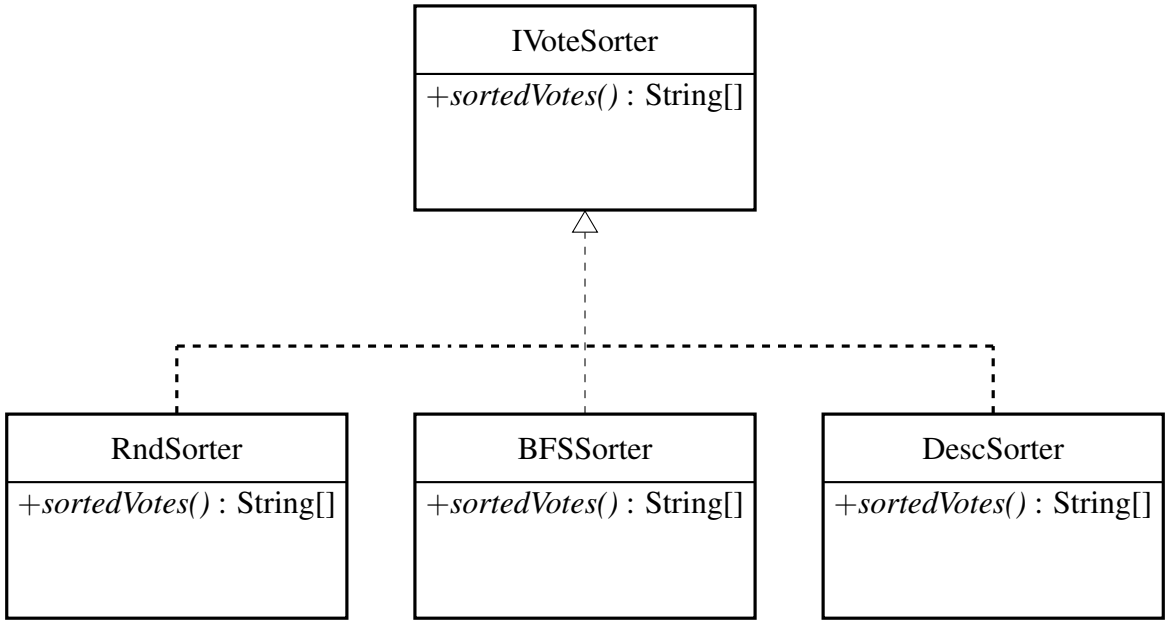

Figure 5: Strategy Pattern for vote sorters

5 Results

	Min time	Max time	Average time
Algorand	758 s	2149 s	939 s
Bitcoin	796 s	1512 s	1038 s
Ethereum	711 s	2243 s	1108 s

Table 2: INSERT CAPTION

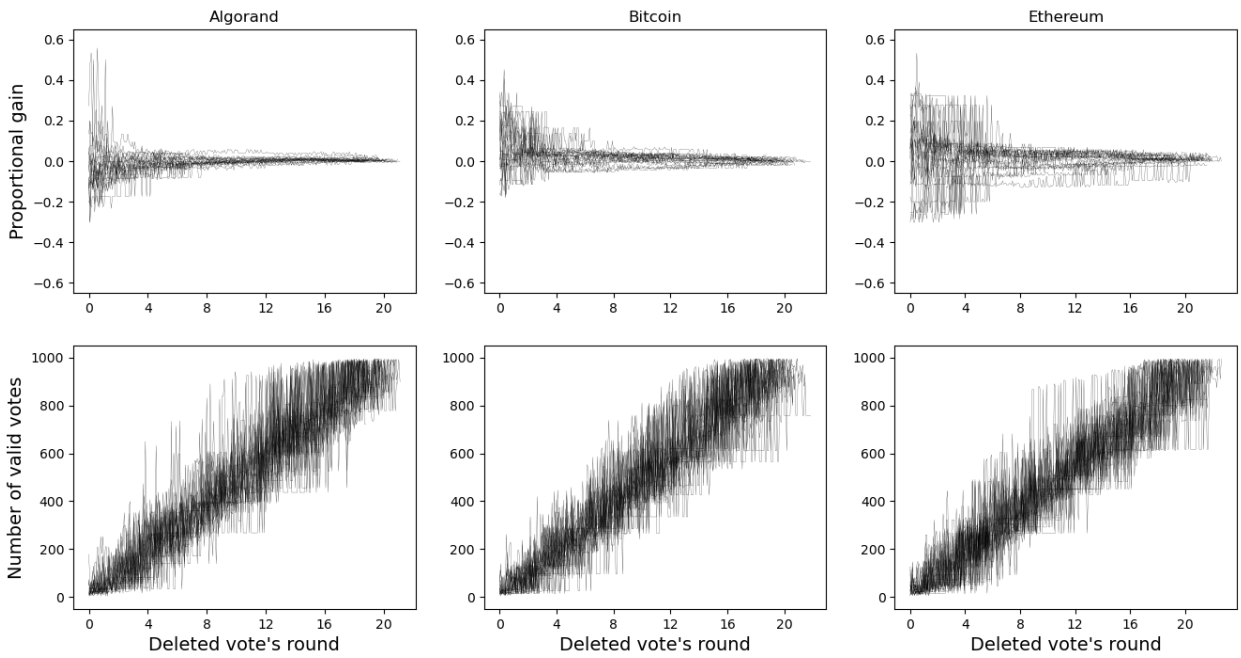

Figure 6: INSERT CAPTION

6 Discussion

7 Conclusion

8 Appendix

Here is the source file for the primary data structure: the Tangle. If you would like to see the rest of the code base, I can give you access to the repository. I chose not to include more files because I think the source code amounts to about 50 pages of text.

```
package tangly;

import java.io.*;
import java.util.Hashtable;
import java.util.ArrayList;
import java.util.Set;
import java.util.Arrays;

import tangly.util.*;
import static tangly.util.Functions.*;
```

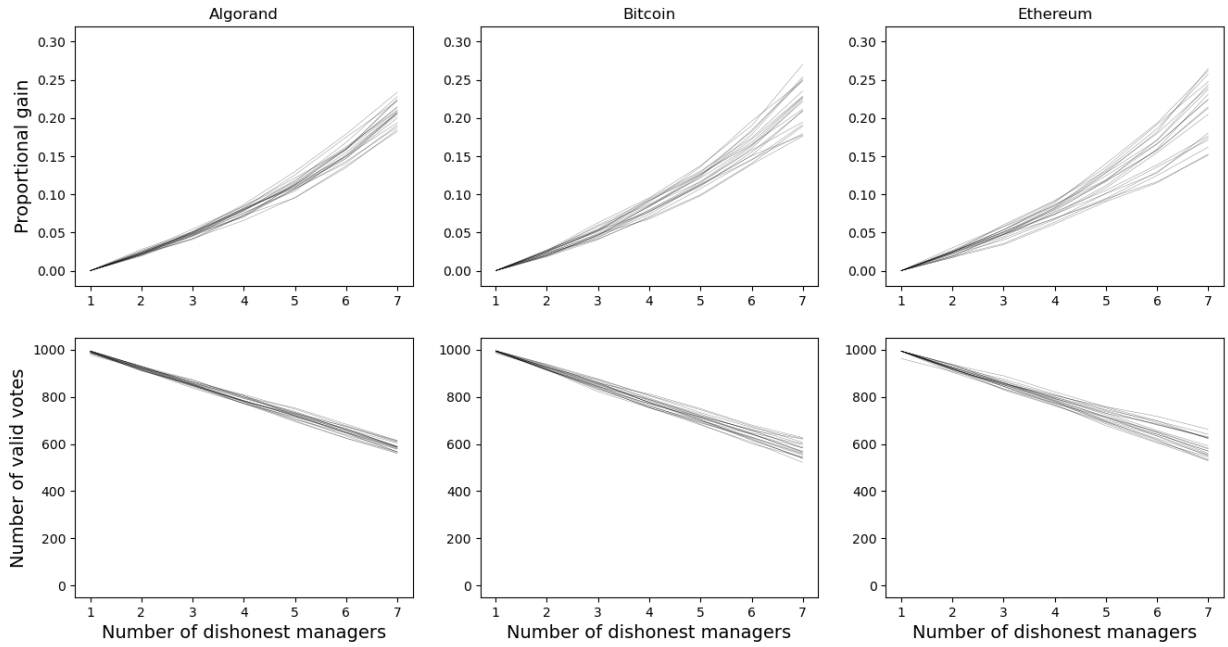

Figure 7: INSERT CAPTION

```
public class Tangle implements Serializable {

// STATIC VARIABLES

private static String ROOT0 =
        bytesToHexString(sha256("00000".getBytes()));
private static String ROOT1 =
        bytesToHexString(sha256("11111".getBytes()));

// CLASS VARIABLES

private String tangle_loc;
private Hashtable<String, Vertex> ht;
private ArrayList<String> tips;

// CONSTRUCTOR

public Tangle(String tangle_loc) {
 this.tangle_loc = tangle_loc;
```

```
 this.ht = new Hashtable<String, Vertex>();
 this.tips = new ArrayList<String>();
 this.tips.add(Tangle.ROOT0);
 this.tips.add(Tangle.ROOT1);
}

// static method to return a deserialized version of the tangle
// should be used by someone who has a serialized version of a tangle
// and needs to instantiate tangle vote in memory. This could be used
// by a DHT node recieving a tangle from another node or a
// deciding node recieving tangles from the DHT
public static Tangle deserialize(byte[] serialized) {
 try {
  ByteArrayInputStream bis = new ByteArrayInputStream(serialized);
  ObjectInputStream ois = new ObjectInputStream(bis);
  Tangle recon = (Tangle) ois.readObject();
  return recon;
 } catch(IOException e){
  System.err.println("caught exception: " + e);
 } catch(ClassNotFoundException e){
  System.err.println("caught exception: " + e);
 }
 return null;
}

// PUBLIC METHODS

// method to serialize tangle for transmission
public byte[] serialize() {
 try{
  ByteArrayOutputStream bos = new ByteArrayOutputStream();
  ObjectOutputStream oos = new ObjectOutputStream(bos);
  oos.writeObject(this);
  oos.close();
  return bos.toByteArray();
 } catch(FileNotFoundException e){
```

```
  System.err.println("caught exception: " + e);
 } catch(IOException e){
  System.err.println("caught exception: " + e);
 }
 return new byte[0];
}

// method to insert a vote into the tangle
public void insert(Vertex v) {
 this.ht.put(v.identifierH(), v);
 this.tips.add(v.identifierH());
 if (this.tips.size() > 2) { this.tips.remove(v.parent1H()); }
 if (this.tips.size() > 2) { this.tips.remove(v.parent2H()); }
}

// method to return the next two tips that should be used
public String[] nextTips() {
 return new String[] { new String(this.tips.get(0)),
                       new String(this.tips.get(1)) };
}

// method to mark the vote at key as invalid
public void markInvalid(String key) {
 if (this.contains(key)) { this.ht.get(key).vote().markInvalid(); }
 else { System.err.println("No such key in hash table"); }
}

// method to mark the vote at key as deleted
public void markDeleted(String key) {
 if (this.contains(key)) {
 this.ht.get(key).vote().markInvalid();
 this.ht.get(key).vote().markDeleted();
}
 else { System.err.println("No such key in hash table"); }
}
```

```
// method to return whether the desired vote is in the tangle
public boolean contains(String key) {
 if (key.equals(Tangle.ROOT0) || key.equals(Tangle.ROOT1)) {
  return true;
 }
 if (this.ht.containsKey(key)) {
  if (!this.ht.get(key).vote().isMarkedDeleted()) {
   return true;
  }
  else { return false; }
 }
 else { return false; }
}

// method to return whether the desired vote is in the tangle
public boolean contains(Vote vote) {
 return this.contains(vote.identifierH());
}

// method to return whether the desired vertex is in the tangle
public boolean contains(Vertex vert) {
 return this.contains(vert.identifierH());
}

// method to populate the children of each vote with the
// reverse of the parents
public void populateChildren() {
 this.clearChildren();
 for (String child : this.keySet()) {
  Vertex v    = this.ht.get(child);
  String p1 = v.parent1H();
  String p2 = v.parent2H();
  if (!p1.equals(Tangle.ROOT0) && !p1.equals(Tangle.ROOT1)) {
   this.ht.get(p1).addChild(child);
  }
  if (!p2.equals(Tangle.ROOT0) && !p2.equals(Tangle.ROOT1)) {
```

```
   this.ht.get(p2).addChild(child);
  }
 }
}

// method to return the keys of all votes that have been marked as invalid
public ArrayList<String> invalidKeys() {
 ArrayList<String> invalid = new ArrayList<String>();
 for (String key : this.keySet()) {
  if (this.ht.get(key).vote().isMarkedInvalid()) { invalid.add(key); }
 }
 return invalid;
}

public ArrayList<String> validKeys() {
 ArrayList<String> valid = new ArrayList<String>();
 for (String key : this.keySet()) {
  if (!this.ht.get(key).vote().isMarkedInvalid()) { valid.add(key); }
 }
 return valid;
}

// PRIVATE METHODS

// method to clear the children of each vote
private void clearChildren() {
 Set<String> keys = this.keySet();
 for (String key : keys) { this.ht.get(key).clearChildren(); }
}

// method to return the number of votes in given round in the ancestry of th
public int numVotesInRound(int num_rnds, int p_round, String[] seeds) {
 int counter = 0;
 Hashtable<String, Boolean> added = new Hashtable<String, Boolean>();
 ArrayList<String> queue = new ArrayList<String>();
 for (int s = 0; s < seeds.length; s++) {
```

```
  queue.add(seeds[s]);
  added.put(seeds[s], true);
 }
 for (int q = 0; q < queue.size(); q++) {
  String key = queue.get(q);
  if (!key.equals(Tangle.ROOT0) && !key.equals(Tangle.ROOT1)) {
   Vertex vert = this.ht.get(key);
   int rnd = round(num_rnds, this.tangle_loc, vert.vote());
   if (rnd == p_round) { counter++; }
   if (!added.containsKey(vert.parent1H())) {
    queue.add(vert.parent1H());
    added.put(vert.parent1H(), true);
   }
   if (!added.containsKey(vert.parent2H())) {
    queue.add(vert.parent2H());
    added.put(vert.parent2H(), true);
   }
  }
 }
 return counter;
}

// method that returns whether there are enough votes in the
// ancestry of the seeds to satisfy the min_votes requirement
public boolean atLeastMinVotes(int num_rnds, int min_votes,
                               int p_round, String[] seeds) {
 int counter = 0;
 Hashtable<String, Boolean> added = new Hashtable<String, Boolean>();
 ArrayList<String> queue = new ArrayList<String>();
 for (int s = 0; s < seeds.length; s++) {
  queue.add(seeds[s]);
  added.put(seeds[s], true);
 }
 for (int q = 0; q < queue.size(); q++) {
  String key = queue.get(q);
  if (!key.equals(Tangle.ROOT0) && !key.equals(Tangle.ROOT1)) {
```

```
   Vertex vert = this.ht.get(key);
   int rnd = round(num_rnds, this.tangle_loc, vert.vote());
   if (rnd == p_round) { counter++; }
   if (!added.containsKey(vert.parent1H())) {
    queue.add(vert.parent1H());
    added.put(vert.parent1H(), true);
   }
   if (!added.containsKey(vert.parent2H())) {
    queue.add(vert.parent2H());
    added.put(vert.parent2H(), true);
   }
  }
  if (counter >= min_votes) { return true; }
 }
 return (counter >= min_votes);
}

// GETTER METHODS

public Set<String> keySet()   { return this.ht.keySet(); }
public Vertex get(String key) { return this.ht.get(key); }
public String getTangleLoc()  { return this.tangle_loc; }

// STRING METHODS

public String toString() {
 String ret = new String(this.tangle_loc + "\n");
 for (String key : this.keySet()) {
  ret += this.ht.get(key).toString();
 }
 return ret;
}

}
```

References

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[2] Wei Cai, Zehua Wang, Jason B Ernst, Zhen Hong, Chen Feng, and Victor CM Leung. Decentralized applications: The blockchain-empowered software system. *IEEE Access*, 6:53019–53033, 2018.

[3] Asad Ali Siyal, Aisha Zahid Junejo, Muhammad Zawish, Kainat Ahmed, Aiman Khalil, and Georgia Soursou. Applications of blockchain technology in medicine and healthcare: Challenges and future perspectives. *Cryptography*, 3(1):3, 2019.

[4] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.

[5] Iuon-Chang Lin and Tzu-Chun Liao. A survey of blockchain security issues and challenges. *IJ Network Security*, 19(5):653–659, 2017.

[6] Victor R Basili, Gianluigi Caldiera, and H Dieter Rombach. The goal question metric approach. *Encyclopedia of software engineering*, pages 528–532, 1994.

[7] A. Palai, M. Vora, and A. Shah. Empowering light nodes in blockchains with block summarization. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, 2018.

[8] Ulfah Nadiya, Kusprasapta Mutijarsa, and Cahyo Y Rizqi. Block summarization and compression in bitcoin blockchain. In *2018 International Symposium on Electronics and Smart Devices (ISESD)*, pages 1–4. IEEE, 2018.

[9] J. D. Bruce. The mini-blockchain scheme (a.k.a purely p2p crypto-currency with finite mini-blockchain), Jul 2014. Accessed: 2014-04-05.

[10] A. Marsalek, T. Zefferer, E. Fasllija, and D. Ziegler. Tackling data inefficiency: Compressing the bitcoin blockchain. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 626–633, 2019.

[11] Roman Matzutt, Benedikt Kalde, Jan Pennekamp, Arthur Drichel, Martin Henze, and Klaus Wehrle. How to securely prune bitcoin's blockchain. *ArXiv*, abs/2004.06911, 2020.

[12] Serguei Popov. The tangle. *cit. on*, page 131, 2016.