Verity Ledger: A Protocol for Improving Data Quality and Ensuring Data Authenticity in Publicly-Built Open Datasets

by

Israel R. Macias

S.B., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

Author .....
Department of Electrical Engineering and Computer Science
May 12, 2020

Certified by .....
Miho Mazeereuw
Associate Professor of Architecture
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Verity Ledger: A Protocol for Improving Data Quality and Ensuring Data Authenticity in Publicly-Built Open Datasets

by

Israel R. Macias

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2020, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In recent years, there has been a heightened interest in the sharing economy for open data. One example is the Open Street Map (OSM) where people contribute to open repositories of street data for the public to use. These datasets are rich as a result of being constructed from data that the public created, rather than static information such as fact data. OSM is a system that relies on public contributions to continue to build and update its map, but therein lies a few critical problems.

For one, there is no real incentive for the public to contribute aside from good will. This limits the size that the data set can grow to and its usefulness. Second, for these datasets (and even publicly available data in general), anyone who uses the data must trust the entity that provides it. This introduces an authenticity problem. There is no guarantee that the data owner is trustworthy. Lastly, because of the reliance on truthful contributors, there is no guarantee that the data is of good quality, posing an issue for those using the data.

The Verity Ledger is a decentralized solution to these problems and is built as a ledger of database transactions with validators and contributors participating in data exchange. Based on an Open Representative Voting (ORV) protocol for consensus, Verity is constructed with each of the aforementioned public dataset issues in mind, using the ORV protocol to solve each of them. This thesis presents the overall architecture of the system and how they are integrated together to function in a decentralized manner.

Thesis Supervisor: Miho Mazeereuw
Title: Associate Professor of Architecture

Acknowledgments

These past 5 years have been quite the ride, and there have been many people that have supported me across the years. For starters, I would like to thank my previous lab group at the RLE Speech Communications Group. Working under Elizabeth Choi was an absolute pleasure, and I was able to learn a lot about speech recognition and ML techniques during my time there.

Additionally, thank you to my research group at the MIT Urban Risk Lab for allowing me the opportunity to work with them throughout my final two years at MIT. They all have a clear mission to help the world at large, and have been wonderful people to work with. Miho for being a supportive leader for the team, Aditya for always checking up and offering helpful advice, Mayank for his guidance and support on the RiskMap/FEMA projects, Saeko for her hard work and many meetings with people across Japan to support RiskMap, Evan for his technical guidance and advice, Larissa for her excellent work with the FEMA projects, and David for keeping the space lively and well.

To those who have offered their advice and suggestions and how to improve this system design, thank you. I would also like to thank the MIT Sport Taekwondo team for all of the strong bonds we were able to form as a team. I always looked forward to sparring and competing in tournaments with such incredible people. Practicing and interacting with the team always turned a stressful day to a happy one. To my friends and those in the MISTI Japan program, thank you for being awesome and supportive. Thank you to Christine Pilcavage for helping me discover the Urban Risk Lab, and being an awesome program director for the MISTI Japan program.

Lastly, I would like to thank my family for their support even as the world deals with COVID-19. Whatever the circumstance, they have always been incredibly supportive. To my older brother, thank you for always offering all kinds of advice whenever I needed it.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to Blockchain

Decentralized ledger technologies, more commonly known as blockchain technologies, have recently been gaining massive attention due to their potential of revolutionizing transaction-based systems.

A **blockchain** is an append-only data structure that holds a data object, known as a block, that are strung together [18]. Another similar data structure is the *singly linked-list* which holds only one-way pointers from an object to the next object. The main difference here is that a blockchain is append-only, which means that data can only be *added* and never removed from the chain. How they revolutionize transaction-based systems is that they can act as a ledger of transaction history, allowing anyone who has access to the ledger to be able to build the state of a system based on said history.

Achieving decentralization of a transaction-based system with a blockchain requires making use of a P2P network, where participants in the network, through some consensus model, can all agree on the state of the blockchain ledger [23]. If the participants are able to maintain a truthful ledger without giving power to any single network participant (distributed, decentralized network), then decentralization is possible. This allows transaction-based systems such as payments to exist without the need of a central bank [18]. An additional component of decentralized blockchain

systems is that they offer *anonymity* to the transactors, a feature not possible with centralized banks [18].

Maintaining truth in a blockchain relies on transaction verifiers known as **miners**, who work to ensure that transactions are valid. By distributing the verification task across many miners, network attacks such as the double-spending problem and Sybil attack vulnerabilities common to previous digital cash solutions are resolved, and with an append-only distributed ledger, we can guarantee authenticity, validity, and trust [18].

However, these systems rely heavily on participation from public miners/validators in order to healthily distribute the network hash rate and prevent these 51% attacks. For this reason, it is difficult to produce a successful blockchain without a strong use case, leading to many failed blockchain projects [2].

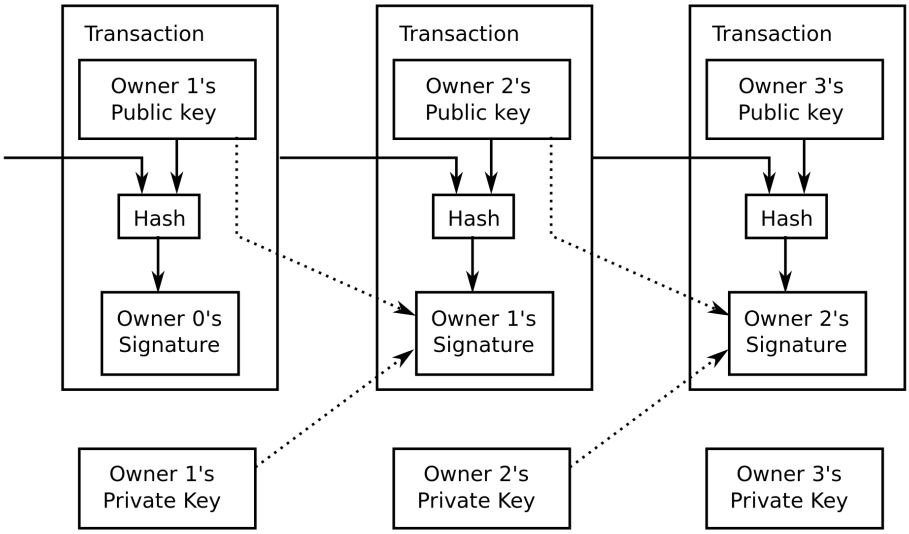

Figure 1-1: Blockchain Data Structure first proposed by Nakamoto in Bitcoin. Blocks in the blockchain are transactions are ordered from left to right in the ledger.

The first and largest of these systems is Satoshi Nakamoto's Bitcoin blockchain, which today has amassed a network hash rate of over 100 million TH/s with an overall market cap of above $125 billion [1]. In its core, Bitcoin is a solution to payments and introduced digital cash to the world with a large perceived value. Several other

parties have followed suit to develop blockchains for other or similar use cases that may seem appropriate for blockchain. Some examples include smart contracts, IoT, and privacy, leading to the overall creation of over 1,500 digital currencies [1].

1.2 Decentralization for Public Data

In some instances, there have been questionable use cases for blockchain, but there are also scenarios where blockchain can indeed serve a purpose. Blockchain can support publicly-built datasets, that is, datasets where the public can freely contribute, is one such use case. Contributions to these datasets yields no benefit to the contributor and often times there are not any strong mechanisms to verify the quality of the data that contributors submit. Additionally, when it comes to appending data, these datasets are often times not secure, leading to questionable authenticity. A dataset owner can easily manipulate the data an individual submits, unless an authentication scheme through means such as Google or Facebook are used, which forfeits anonymity.

An example of a public dataset that exhibits these qualities is OpenStreetMap where volume of contributions is small and authenticity is not guaranteed [19]. OpenStreetMap is a publicly-built map that relies on data supplied by the public to maintain the truth of the map. However, a strict standard for verification is not present, and without enough contributors, there are many holes in the map [19]. Wikipedia is another example, which although has a high volume of contributions, originally suffered from the loose verification constraints, leading to false data being submitted to Wikipedia pages [5].

By leveraging decentralized ledger technologies, these problems could be dealt with, provided that the correct system is architected. If the system could promote contributions through incentives, while having a verification/validation mechanism in place, then the challenges these datasets face can be mitigated.

1.3 The Verity Ledger

For this reason, we propose the Verity Ledger, an immutable distributed ledger aimed at providing authenticity for permissioned and permissionless datasets, alongside reducing the problems of validity and potentially increasing public contribution volume if needed. This ledger's main purpose is to support open datasets rather than replace them, and would exist as a lightweight ledger with metadata that the public can use to ensure that the data they read is of good quality, and authentic.

Throughout this thesis we explore several consensus protocols that can be used with Verity, its overall system architecture, and its use cases across public datasets.

Chapter 2

Related Works

2.1 Direct Acyclic Graphs

Decentralized transactions sparked interest in developing a number of specialized blockchains, each with their own specific consensus protocol in mind. Consensus is possibly the largest factor in determining how a decentralized system works when faced with conflicting transactions [7]. However, the data structure used to store and propagate the ledger is just as important [23]. Although blockchain, the most common data structure that is used, is suitable to uphold the necessary features of a decentralized network, it fails with regards to performance and efficiency [4]. Namely,

1. Transaction throughput
2. Ledger size
3. Efficiency

An emerging blockchain substitute to help deal with each of these issues is a directed-acyclic-graph (DAG), which is in essence a graph of transaction nodes pointing to one another without cycles, dissolving the double spend problem [4]. Two popular DAG-based ledgers are Nano's block-lattice technology [17] and the IOTA tangle [21].

2.1.1 Block-lattice in Nano (Raiblocks)

One of the most well-known DAG-based distributed ledgers is known as Nano (previously Raiblocks), which relies on a block-lattice to support faster transactions and increased throughput through the blockchain. A **block lattice** is a a type of DAG ledger architecture where each account possesses their own blockchain [17].

A typical blockchain ledger would store all transaction history (after regular pruning), resulting in a ledger that increases in size linear to the number of transactions, severely throttling throughput and thus transaction speed. By distributing the ledger to all accounts rather than consolidating it into a single ledger, one problem is solved. To further reduce ledger size, each block-lattice account stores *only* an account balance, and issues transaction based on a representative system, **asynchronously**.

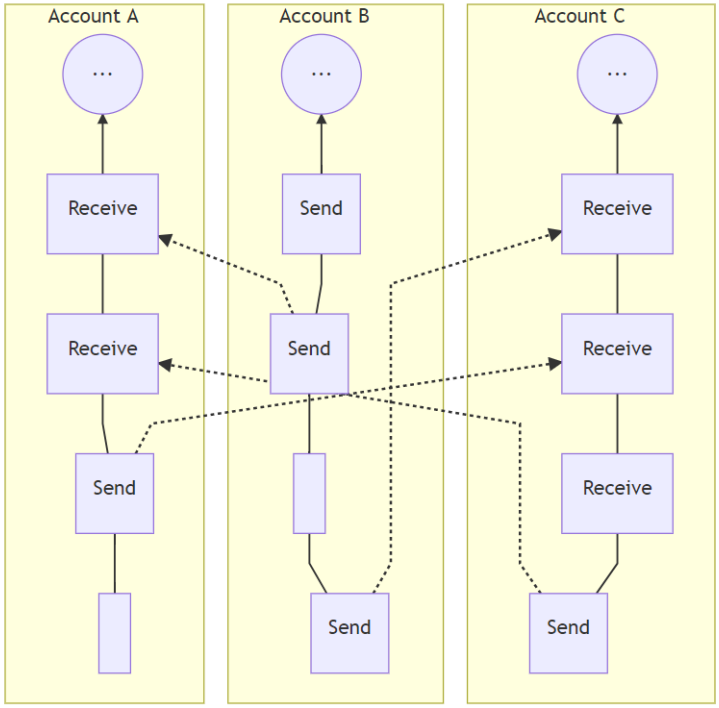

Figure 2-1: Nano Account Ledger. Each account stores only an account balance for increased scalability and can issue either send or receive transactions asynchronously from the rest of the network.

Proof-of-work in Nano is done through issuing two transaction types, **receive** and

send. Thus, to issue a transaction between two accounts, a sending account needs to issue a send transaction to a receiving account that issues a receiving transaction, fulfilling the transaction contract [17].

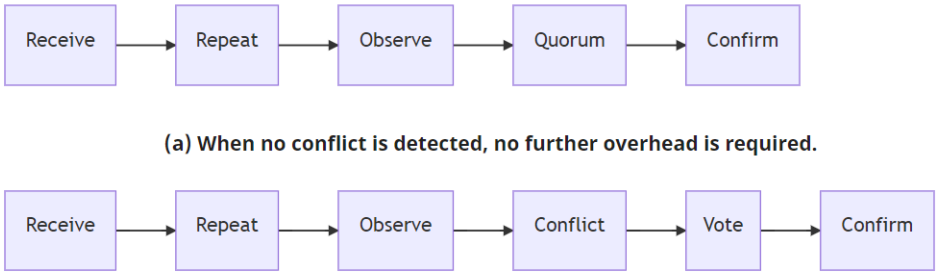

Figure 2-2: Nano Transaction

For transaction verification, each account must designate a trusted representative to vote on its behalf to the network. This representative uses the block information provided by the transaction issuers along with its voting weight to reach a decision and ultimately decide whether to confirm the transaction or not, which is discussed in more detail in Section 3.2.1.

2.1.2 The IOTA Tangle

Another DAG-based distributed ledger is known as IOTA, whose data structure is a DAG of transactions known as a *tangle* rather than accounts [21]. IOTA is a distributed ledger platform focused on microtransactions for IOT devices that allows them to issue micro data requests at high scalability and throughput. The main difference in the IOTA platform is that transaction verification is done by the transaction issuers rather than dedicated miners.

The tangle starts first with a genesis node (transaction) that begins the DAG, and is then followed up by new transactions. As transactions are issued, they are appended to the DAG by adding an edge from two graph leaves (known as **tips**) to the new transaction node. An edge in the tangle from node $A \rightarrow B$ means that A is confirmed by B. For a node to be registered in the network, two confirmations

(and thus two edges) are required from any tip to a new transaction. When there are multiple tips in the tangle (at least one is guaranteed to be in the network), a **tip selection** algorithm is run in order to select the best two tips to confirm. The best two tips here just refers to transactions that either have been hanging for a long time, or are moving a large sum of an account balance. This determines the weight that each transaction node has.

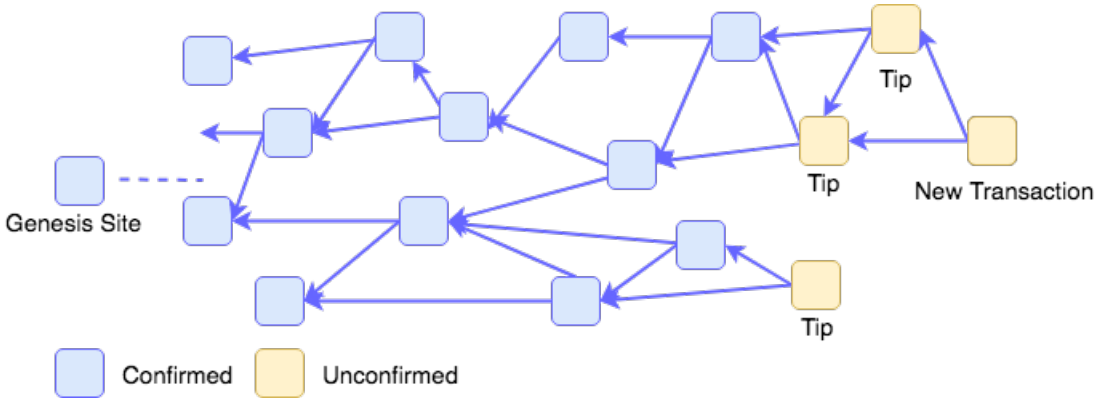

Figure 2-3: IOTA tangle with genesis transaction and confirmation flow. New transactions must choose two tips to confirm, requiring a small amount of Proof-of-work confirmation.

Tip selection in IOTA is a rather complicated process, but can be summed up with the following pseudo-code:

Result: tip node

select N random sites in the tangle;

for *site in N* **do**

    start_random_walk(site);

end

if *site reached tip and meets time thresholds* **then**

    return as tip node;

end

Algorithm 1: Simple Tip Selection in IOTA

Here, new transactions use a **MCMC** (Markov Chain Monte Carlo) algorithm to sample two tips by randomly walking through random sites in the tangle, and returning once two valid tips have been found [22]. Nodes are weighted according to

a number of factors to facilitate the random walks. After finally locating two tips, the transaction can be appended.

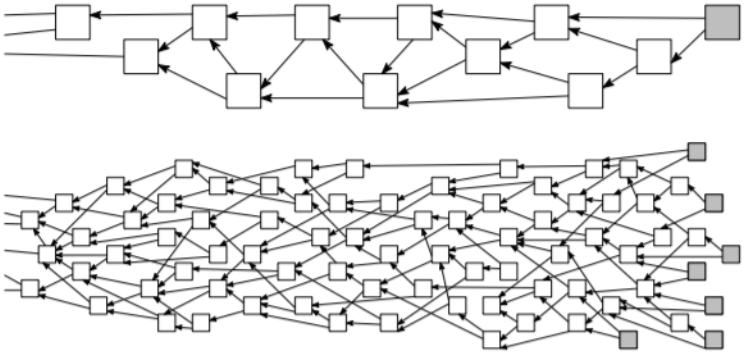

Figure 2-4: The IOTA Tangle as shown in the whitepaper. The top DAG is a low-load tangle network while the bottom tangle refers to a high-load tangle.

Because new transactions must select two tips to confirm before being attached to the network, it is theoretically expected that the system performs **faster at a higher load**. This is because of the $2:1$ ratio of confirmations $\rightarrow$ new transactions upon network usage. In practice, this is of course much different, mainly because the tangle has yet to mature to view its full performance [22].

Similar to Nano, IOTA offers increased transaction throughput while moving the Proof-of-Work computation off to the transaction issuers. Both of these DAG-based systems result in significant improvements over Bitcoin's blockchain, while providing a platform that is conducive to their own specific purpose,

2.2 Ethereum

Ethereum is one of the earliest ledgers in the blockchain space and gained a large following due to its Proof-of-Stake (PoS) and Distributed Proof-of-Stake (DPoS) consensus algorithms, the introduction of *Smart Contracts*, and its general performance improvements when compared to Bitcoin [25].

2.2.1 Proof of Stake Algorithms

Traditional Proof-of-work systems rely on miners in order to append a block to a blockchain in exchange for a small reward, typically at the cost of expensive GPU hardware and high electricity use [24]. Proof-of-Stake is a class of consensus algorithms where instead of rewarding participants based on mining power, they are rewarded as a proportion of their stake in the network [25]. These participants are validators who vote in a *quorum* when approving a new transaction, reducing the cost of block approvals, since validators do not need to solve computationally expensive cryptographic puzzles for validation.

2.2.2 Smart Contracts

A **smart contract** in a distributed ledger is a function or protocol that facilitates an agreement between two parties [8]. In general, two parties would typically agree on a transfer amount (in a blockchain's native currency) and execute based on agreed-upon conditions. The advantage that smart contracts have over "physical" contracts is that they are controlled by a third party in a decentralized manner. This third-party, acts as an arbiter, that executes the contract's functions once certain conditions have been met. Typically these contracts deal with value transfer between two entities, and only complete the transfer of value once the smart contract is fulfilled. In Ethereum, the contract is stored in the network and makes it *public* for anyone to read, but impossible to determine who it was between [8].

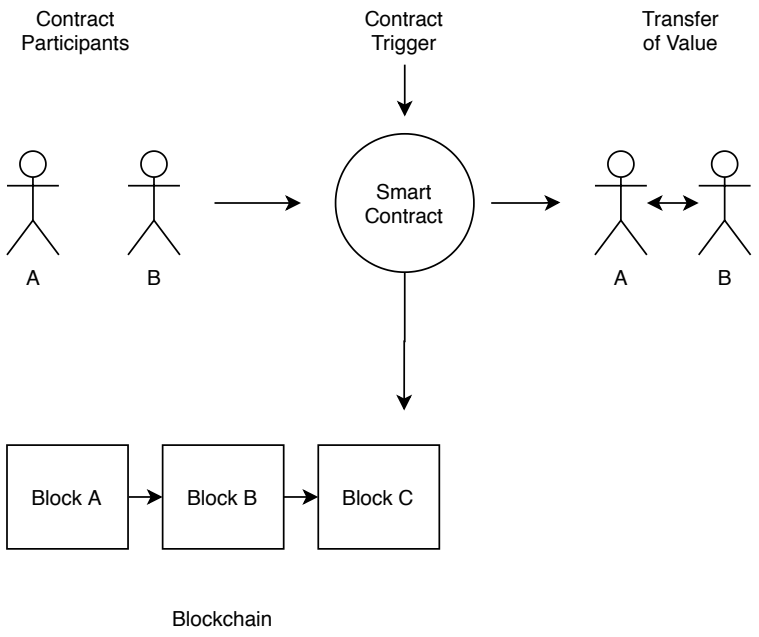

Figure 2-5: Smart Contract Flow. A Smart Contract is recorded in the blockchain and contains criteria that must be met in order to execute. Once it has executed, the transfer of value between the contract's participants is completed.

Because the identity of those forming the smart contract never needs to be disclosed, a transfer of value between two parties can occur anonymously without needing to sacrifice trust, preserving anonymity, autonomy, and allowing the contract to be backed up in the blockchain's history. Most new blockchain/distributed ledgers rely on smart contracts in order to function for their specific use case. The Verity ledger also relies on a smart contract between the public and the data nodes describe in Section 4-1.

Chapter 3

Consensus Protocols

3.1 Overview of Consensus Mechanisms

A blockchain consensus mechanism has the task of verifying transactions fairly in a Peer-to-Peer environment. With no central authority handling transaction verification, there must be a standard protocol through which an entire network can agree on the state of the ledger, while being resilient against network attacks [23]. There are several consensus mechanisms currently implemented to handle transaction verification such as Proof-of-Work (PoW), Proof-of-Stake (PoS), Proof-of-Authority (PoA), and Open Representative Voting (ORV) [7]. They differ based on the way they define their own verification protocol and, each has its own advantages.

Consensus Mechanism	Proof-of-Work	Proof-of-Stake	Proof-of-Authority	Open Representative Voting
Probability of Successfully Adding a Block	Increases as Computation Increases (reward scales with input)	Increases linearly with the number of network tokens/coins held (stake)	Increases based on an account's network reputation	Proportional based on voting weight
Throughput	Low	High	High	High
Energy Costs	High (often requires specialized ASIC miners)	Low (no mining required)	Low (no mining)	Low (work computation is small for anti-spam measures)
Main Weakness	Slow, and susceptible to centralization if few miners control most of the hash rate	Requires Stake to be distributed in a way that does not allow centralization	Requires a good validator approval standard	Can be susceptible to Sybil attacks if balance is not distributed properly.

Table 3.1: Popular Consensus Protocols and their Attributes

3.1.1 Proof-of-Work

Among each of these mechanisms, the most popular is proof-of-work. In Proof-of-Work, transaction verification is done as follows:

1. Each account has a copy of the blockchain.

2. A new transaction between two accounts is added to the chain and must be *confirmed*. This transaction happens at a fee to the transactors.

3. Validators, known as miners, compete to solve a cryptographic puzzle in order to verify that the transaction is valid according to the state of the blockchain.

4. If enough validators have confirmed the transaction, then the transfer of value is completed.

This mechanism is decentralized as many different miners must contribute their own hash power (and thus computational resources) in order to be eligible to receive a reward for mining a block of transactions (block reward) [18]. As the number of different miners increases, the network becomes more and more decentralized. The Bitcoin block reward is valuable, so there is a large incentive to compete, but it requires electricity and expensive hardware [9]. Additionally, there are several critical weaknesses with this protocol, namely wasted work (miners who compete but do not get a block reward waste energy), low speed, and vulnerability to a Sybil attack if the network is not decentralized enough. That is, if any single miner controls most of the hashing power (more than 50%), it has the power to completely rewrite the blockchain and generate false transactions [12]. This is one of the largest reasons why a secure consensus mechanism is required.

3.1.2 Stake-based Consensus

Proof-of-Stake (PoS) is a viable alternative that resolves the environmental impact of PoW systems while maintaining decentralization. With PoS, there are *validators* instead of miners who vote on a transaction's validity with their *voting weight* in exchange for a small reward. A validator's voting weight is based on the amount of currency they possess combined with the age of the validator's currency [25]. This means that a validator who freezes their balance will have a larger stake in the network over time, and can *increase* their balance by validating transactions. Ethereum's Casper is one such protocol that implements a modified PoS mechanism, except that

validators who act maliciously, by voting for false blocks for example, are punished and have their stake reduced [6]. This offers an additional layer of trust, as validators are incentivized to act truthfully.

Another stake-based consensus protocol is Proof-of-Authority (PoA) which is an adapted version of Casper. In PoA stake is not based on the amount of currency owned by a validator, but rather the **reputation** that the validator has built [10]. Defining reputation depends on the system that uses PoA, but is in general a weighted calculation of correct confirmations by the validator. Thus, building a reputation means verifying transactions truthfully in order to receive a block reward. A feature of Proof-of-Authority consensus mechanisms is that only verified authorities can validate data and will stake their reputations upon false validation. That is, the higher a node's reputation, the higher the node's stake or hashing power in the network.

It is generally accepted that the following three characteristics are required for proof-of-authority systems to function:

1. Validators are Trusted

2. Reputation is Valuable

3. Equal Standard for Validator Approval

Proof-of-Authority was originally the best consensus model for Verity. By focusing on *reputation* as a form of stake, it provides the best form of incentive for validators (miners) to act in a non-malicious manner. However, Proof-of-Authority is more useful in *semi-centralized* systems, as validators are chosen by the network owners. In the original proposal for Verity, the main focus was to *incentivize* people to contribute to open datasets and to act as trusted validators in a *decentralized* manner, making Proof-of-Authority a weaker choice.

In this iteration of Verity, however, while incentivizing contributions and validation through some form of blockchain reward is still possible, it is not the priority. The priority is to ensure that any data record that has been submitted to a dataset that is tied to Verity is *authentic* and of good quality while maintaining anonymity and

decentralization. For this purpose, the Open Representative Voting (ORV) consensus protocol is chosen for Verity, the same mechanism used for the Nano Cryptocurrency described in Section 2.1.1 [17]. When discussing the system architecture in more depth in Section 4.1.3, it becomes clearer why ORV makes more sense.

3.2 Verity Ledger Consensus - Open Representative Voting

For Verity Ledger, because no mining will occur, Proof-of-Work will not be used. Instead, validation will occur through Open Representative Voting. In the Verity network, each node owns its own dataset whose data records each contain an additional field: a **transaction hash** which indicates the transaction responsible for inserting the data record. A sample record in a dataset storing survey data of one's income/occupation might look like,

Listing 3.1: Sample Data Record

```
{
    "city": string,
    "occupation": string,
    "income": string,
    "timestamp": UNIX timestamp,
    "transaction_hash": 256-bit integer
}
```

where all of the fields are required information by the dataset administrator and the *transaction_hash* is the unique 256-bit hash of the insert transaction. When someone in the public accesses this record, they can use the transaction hash to look up the state of the record through a block explorer and check items such as the verification state, issued timestamp, representative node, etc. All of these are discussed in more detail when exploring the system architecture.

Note that the ledger does *not* contain the data of the node itself, but rather a

ledger of hashes and required metadata to track history. Because of this, the only true P2P validation will occur across nodes in the network to ensure that no appended data row has been tampered with, and thus ensuring authenticity.

3.2.1 Node Representatives

With this consensus mechanism there are generally two types of nodes that are defined, public nodes and data nodes.

The purpose of **public nodes** is either to validate the *quality* of data records or to contribute to a data node's dataset. Thus they are split between contributor and validator public node types. A data node is a network node that is actively participating in the network by tying their dataset to a representative node, or acting as a representative node itself on behalf of other data nodes that verify the *authenticity* of appended records via a Representative Quorum, much like Nano [17].

For all data nodes, a ledger of append-only transaction history is stored in its account, much like a traditional blockchain. To reduce space consumption as the ledger grows in size, the ledger can be pruned regularly and automatically, and each data node's ledger will reflect this change. There are many techniques for ledger pruning, and can be a component of future work for the Verity system [20]. Non-data nodes are delegates to a representative node, who vote on the behalf of its delegates when verifying transactions in a voting quorum.

Representative data nodes each have their own *voting weight* proportional to a weighted sum of contained verified data records (the number of records correctly voted in favor or against during a quorum) and the number of verified data records in its own dataset, should it have a dataset tied to it. It is calculated by:

$$v_j = \Sigma_{i=1}^{M} t_i + |D|$$

where $|D|$ is the number of verified records in the node's dataset and t_i is a verified transaction weight indicating the number of votes that transaction received for verification. When the network is at an early stage it is relatively easy to amass

a higher share of the voting weight, and therefore if the voting weight is tied to the network size, then early participants in the network can be prevented from having an unfair advantage.

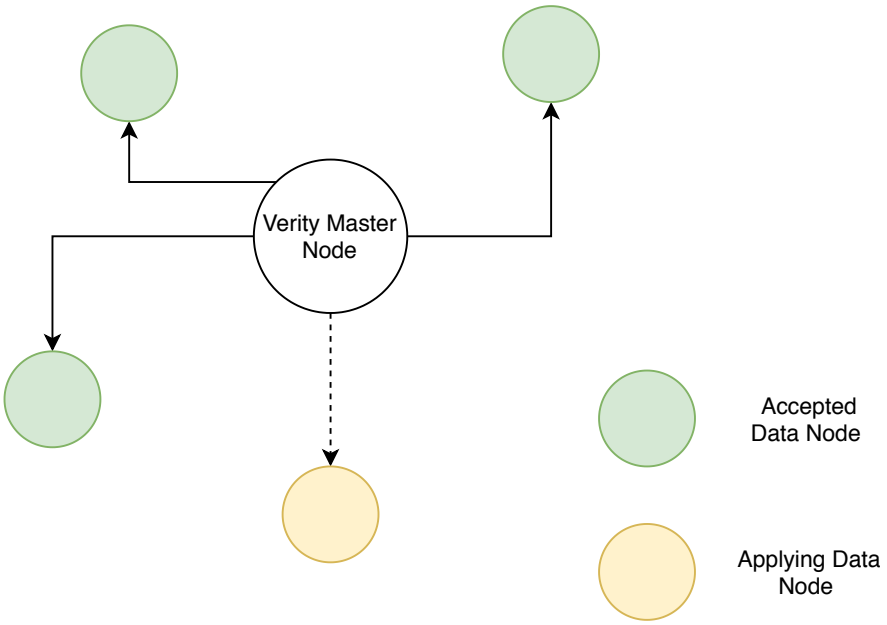

Figure 3-1: Verity Network Conception

Because there are no existing representative nodes at the start of the Ledger's creation, an initial **master node** is created to manage consensus. It handles Verity quorums until enough representatives exist in the network. Once enough representative data node accounts with sufficient combined voting weight is present, the master node can be detached, and the network can become truly decentralized.

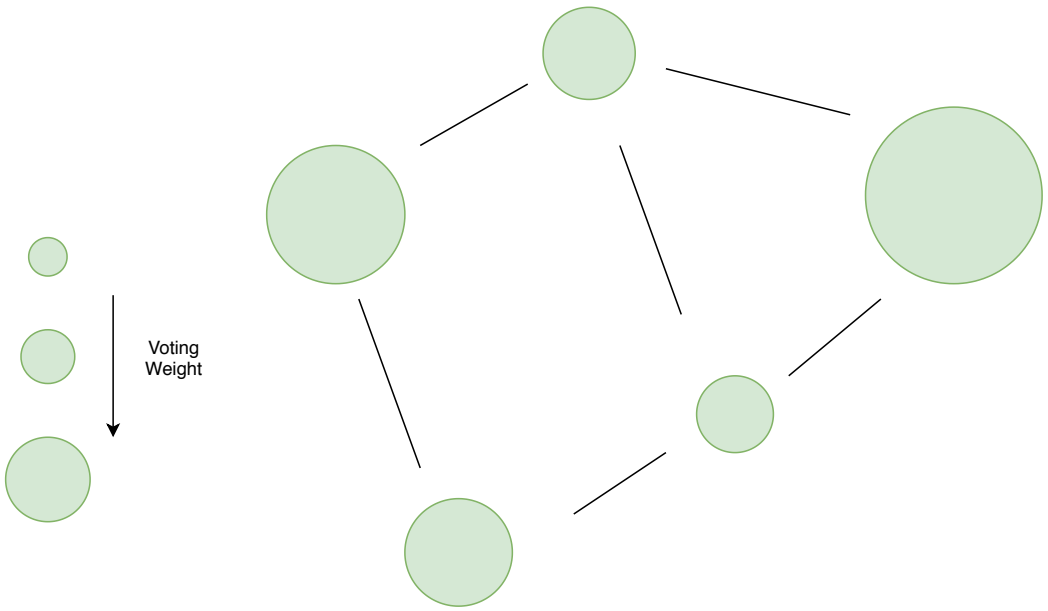

Figure 3-2: Verity Network after Master Node Detachment. A node's size indicates it's voting weight.

To verify a transaction, enough voting weight must be cast in favor of a transaction, requiring some but not all of the nodes to participate. This reduces overhead while speeding up transaction speed. How this is determined depends on the network size, but in general, 50% of network voting weight in favor of a transaction is required to completely verify a transaction.

3.3 Attack Vectors

There are a number of attack vectors that need to be considered when building a decentralized network, the main ones being

1. Sybil Attacks

2. 51% Attacks

Verity is secure against these attacks for the following reasons.

3.3.1 Sybil Attack

A **Sybil attack** occurs when a single entity creates multiple accounts in a decentralized network, each of which can vote in favor of false transactions [11]. Generally, in order to protect against these attacks, a centralized authority is required that can verify the identity of network nodes. However, in P2P networks, when decentralization is required, there is no single trusted authority. For this reason, many early networks had the difficulty of establishing themselves as truly secure networks.

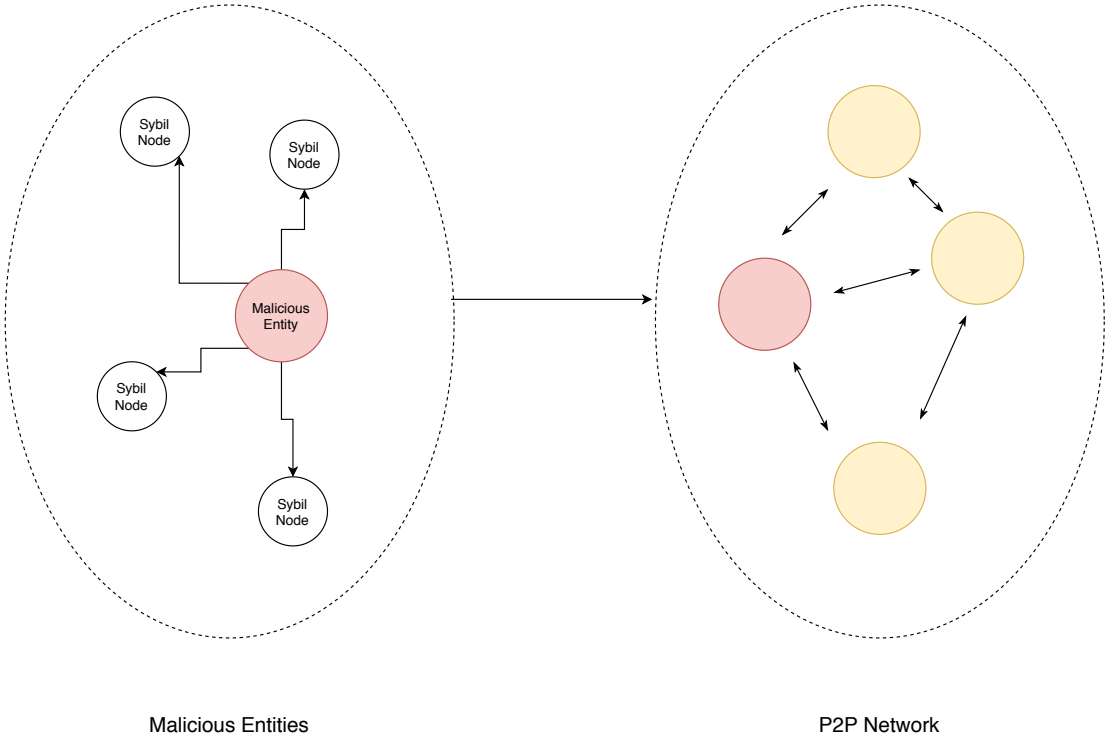

Figure 3-3: Sybil Attack. Malicious entity on left creates multiple fake nodes agreeing on a false transaction, eclipsing an honest P2P node, and causing that node to affect the rest of its network view.

After the development of blockchain through Bitcoin, however, Proof-of-Work systems made Sybil attacks much more difficult to execute, as each malicious need would have to submit a correct Proof-of-Work, requiring computational power rather than simply voting in the network. Verity's Open Representative Voting protocol is resilient against Sybil attacks for this reason [17].

In Verity, if a malicious node exists and generates a false transaction to its Sybil nodes that is broadcasted to the network, they would still fail to undermine authority. All nodes combined in the malicious network would require enough voting weight to influence the network which means that nodes would need:

1. A large enough dataset with enough **verified records.**

2. To have already verified enough transactions to increase voting weight.

The first cannot happen as each honest node would already know that the malicious node and Sybil nodes do not contain verified records leading to disagreement in a network quorum [17]. The second cannot happen as it would require the nodes to submit votes for verification **with enough voting weight** to be trusted, according to our mechanism. For these reasons, a Sybil attack is impractical for anyone to execute and almost impossible given the design of the consensus protocol.

3.3.2 51% Attack

A 51% attack is an extension of the Sybil attack described previously where an entity is able to control multiple nodes that make up a majority of the network voting power, allowing these nodes to completely rewrite a ledger or blockchain [3].

The reasoning for Verity's security against 51% attacks is the same as with Sybil attacks. It is unreasonable and impractical for a node to own greater than 50% of the network voting weight as this would require a large amount of storage and transaction voting weight. While there is no formal proof-of-work done by each node, storage is an added cost that would require some proof of existing and incurred cost. Although not implemented with Verity, adding a hybrid consensus with another protocol could provide even stronger security features. A potential candidate for a secondary protocol could be Proof-of-Storage, which is the basic idea that nodes must provide proof that they store the data they do to verified authorities so that accesses of the data are ensured to uphold data integrity [16]. This makes sense for this system and is worth exploring once the basic architecture matures.

Chapter 4

Verity Ledger

The Verity Ledger combines the flexibility and throughput advantages offered by DAG-based systems with added functionality to secure open datasets. The main purpose of Verity is to support open data sets by offering a decentralized way of ensuring authenticity, integrity, and optional quality verification.

4.1 System Architecture

To fully understand how Verity is built to work, it is important to decompose the system into a number of interconnected layers. Verity can be broken down into four layers with each handling its own specific purpose: *Data*, *Network*, *Consensus*, and *Application*.

Layer	Components
Data	Account/Database Information; Transaction History; Network Metadata; Representative Information
Network	P2P Layer; Handles Send/Receive information across P2P network nodes
Consensus	Proof-of-Authority and Open Representative Voting; Quorum Process
Application	Front-facing application

Table 4.1: System Layers and the Components they Handle.

4.1.1 Data Layer

The data layer is the base of the Verity ledger, and contains all record information describing the state of the network and the nodes. Information about transaction history, validation, and verification are also stored here. For Verity, or any distributed ledger, to function properly, it is necessary to have a layer dedicated to providing the specifications as to how data is stored and distributed.

Verity's data layer itself has several moving parts that will be explained, notably the types of data-storing nodes across the ledger.

Verity Data Nodes

Similar to Nano, Verity avoids replicating its ledger by having each node (counterpart of Nano accounts) have its own *blockchain ledger*, except that instead of storing the transaction history for the specific account's token/coin balance, the ledger is a chain of CRUD transaction history [17]. We call these nodes *data nodes* and are a fundamental component of how Verity operates.

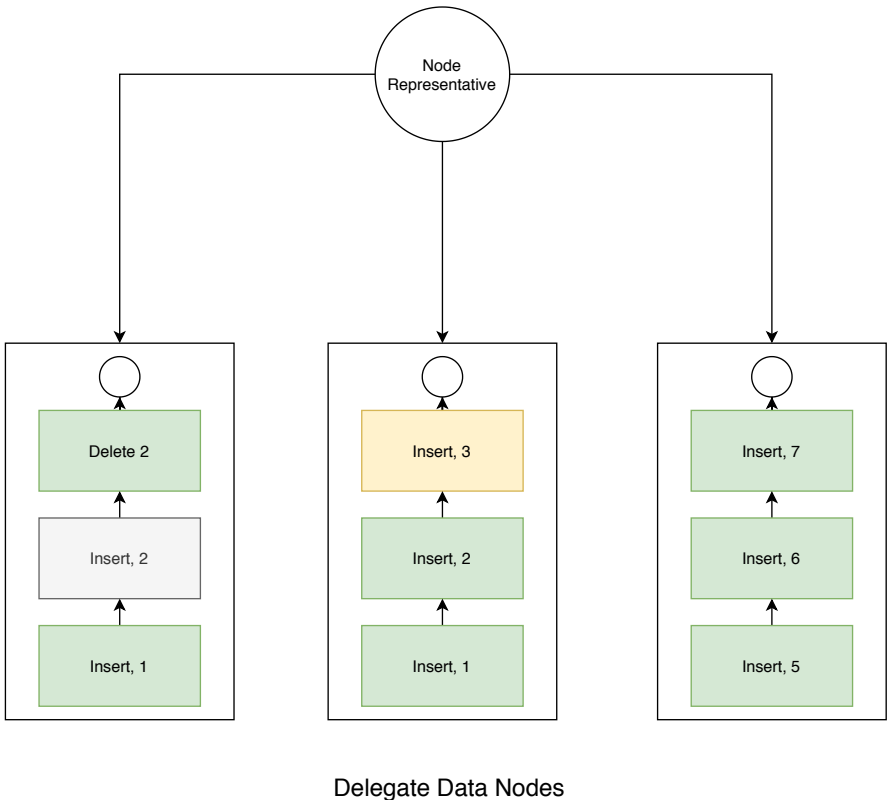

Figure 4-1: Verity Data Representative Node and its Delegate Data Nodes. Each data node contains a ledger of *Insert* or *Delete* transactions and must be the delegate of a representative. Green indicates confirmed transactions, yellow indicates unconfirmed transactions, and grey indicates stale transactions. *Delete* transactions are automatically confirmed, and mark the transaction of the record that was inserted as stale.

The types of CRUD transactions that each ledger can have is reduced to *Insert* and *Delete* for simplicity, although this could be further extended to support additional CRUD transaction types such as *Update*. Because blockchains are append-only data structures, *insert* must be supported; however, the *delete* operation is also allowed should a dataset owner wish to remove low-quality data [13]. In addition to CRUD, Data node ledgers also contain a special transaction known as a *Verify* transaction, which will be further discussed in Section 4.1.3. To be more specific:

1. *data_node.insert*(): A new record has been inserted into the DB. Metadata is logged into the ledger.

2. *data_node.delete*(): An existing record has been deleted from the DB. Metadata regarding the new state of the DB is logged into the ledger (removed insert transaction hash)

At the head of each blockchain/ledger is metadata regarding the state of the DB. Specifically, it is a header of information represented by the following fields:

Listing 4.1: Data Node Metadata

```
{
    "public_address": 256-bit integer,
    "latest_transaction_hash": 256-bit integer,
    "db_page_header": 96 bytes,
    "signature": 256-bit integer,
    "quality_verification": boolean,
    "verification_task": object
}
```

An account is primarily identified by its **account address** which is computed from the account's public and private keys using a cryptographic scheme. For simplicity, this account address can be generated via Bitcoin's address generation procedure described below [18].

Listing 4.2: Basic Address Generation

```
account_private_key = generate_rsa256()
account_public_key = ecdsa(account_private_key).encode('hex')

account_address = sha256(account_public_key.decode('hex'))
```

A 256-bit private key is generated randomly and is then converted into a 512-bit public key via Elliptic Curve DSA [15]. Then, a sha256 base58 encoded address is returned as the public address which the whole network sees. The entire process of address generation is a bit more complicated, but this suffices for the purpose of understanding how node addresses are generated.

Thee *open_quality_verification* boolean specifies whether the data node opted into data quality verification. If the boolean value is set to *true*, then the dataset is exposed to "mining" by *public nodes*. Here, mining refers to a **verification task** and outlines the task that a *validator* must complete to compete for network rewards.

Verity Representative Nodes

Representative Nodes serve the purpose of facilitating consensus. These nodes would typically be trusted data nodes (large nodes with many delegates and 100% data verification) with high perceived data quality.

Representative nodes, in addition to their own ledger should they have one, also maintain a table of **unappended transactions** and their list of delegates (including metadata). These are transactions that have not been verified to be *authentic*, and are only labelled as such once a quorum has been completed.

When a public entity submits data to a data node (adds to an open dataset), they broadcast a *one-way transaction* to the network and to a representative node. Once a data node receives the transaction from the public entity, they broadcast it to their representative node, which hold a **quorum** with other representative nodes to first decide whether or not the transaction is *authentic* (no tampering by the data node) as the representative has the public entity's transaction metadata. If the transaction has been accepted, it will be added to the data node's ledger and can begin awaiting confirmation.

Verity Public Entities

A public entity in Verity is simply an external user/person who submits a CRUD transaction to a data node or a *verify transaction*. They issue one-way transactions as mentioned in 4.1.1 with the record they wish to insert. In addition to this, the same record is sent to **at least 2** representatives in the network for consensus. A public entity can be either a **validator** or a **contributor**. A contributor only needs to *send* data to a public node and does not need an account on the ledger. Each contributor only needs to send a one-time identifying key (address) along with the data they

wish to send to complete an insert transaction. A validator acts as a traditional blockchain miner. Instead of solving computationally expensive puzzles, a validator simply performs a validation task outlined by the data node for potential rewards.

When a public node opens an account in the network, the following fields are populated in the ledger:

Listing 4.3: Public Node Validator

```
{
    "public_address": 256-bit number,
    "representative_address": 256-bit number
    "type": open,
    "balance": 0,
    "signature": 256-bit number
}
```

The validator, like data nodes, must specify a representative for consensus. The consensus protocol for value transfer among data nodes is outlined in 4.1.3 and is similar to Nano's Open Representative Voting System [17].

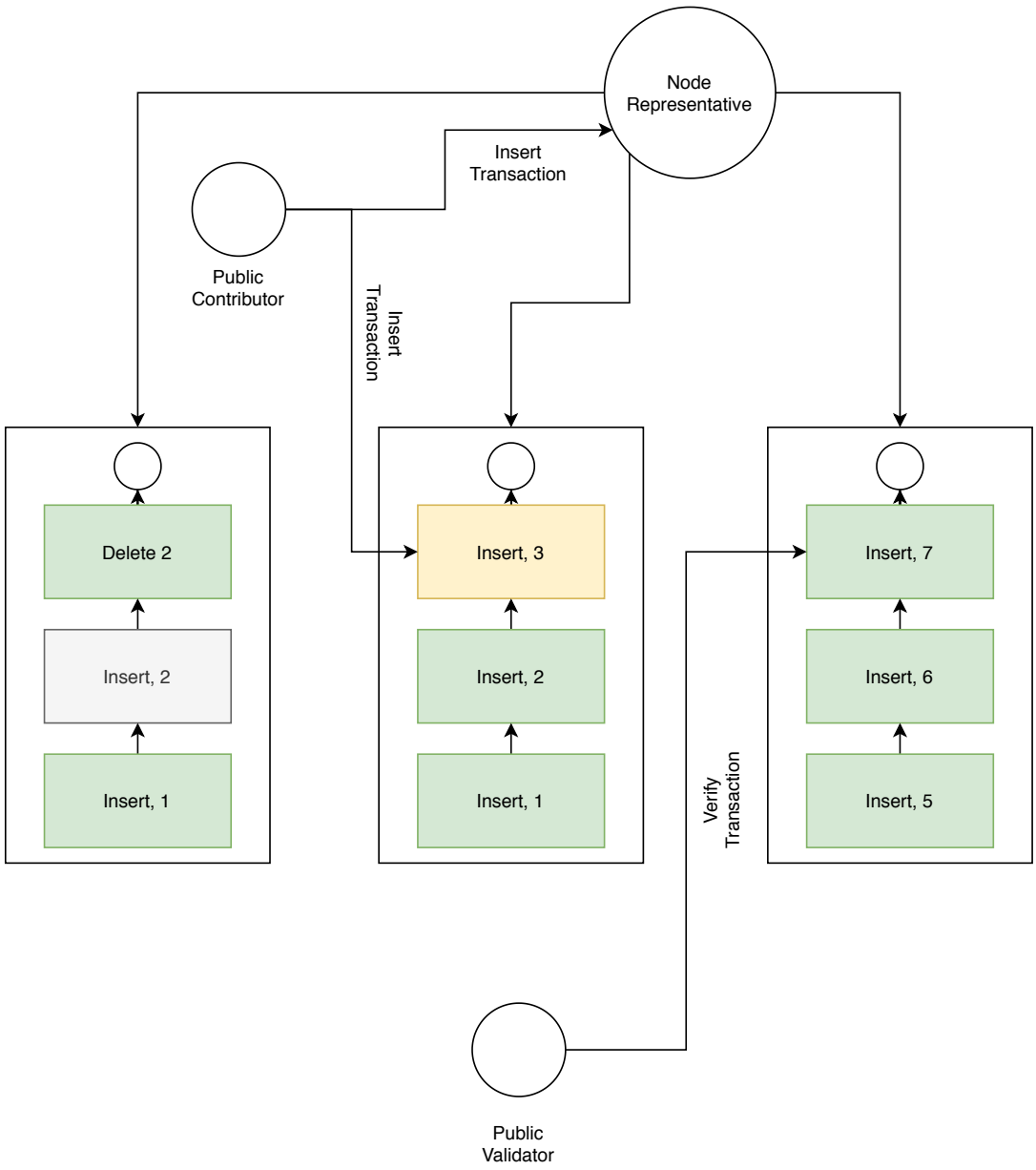

Figure 4-2: Validator and Contributor Public Nodes. The validator successfully validates an unconfirmed insert transaction with hash of 7, marking it *green*. A contributor public node submits data to a data node and a representative, appending it as to the data node's ledger to await confirmation.

4.1.2 Network Layer

Verity, like almost all decentralized systems, operates on a P2P network. The network layer describes how nodes in the network communicate with one another, and how the whole ledger architecture comes together.

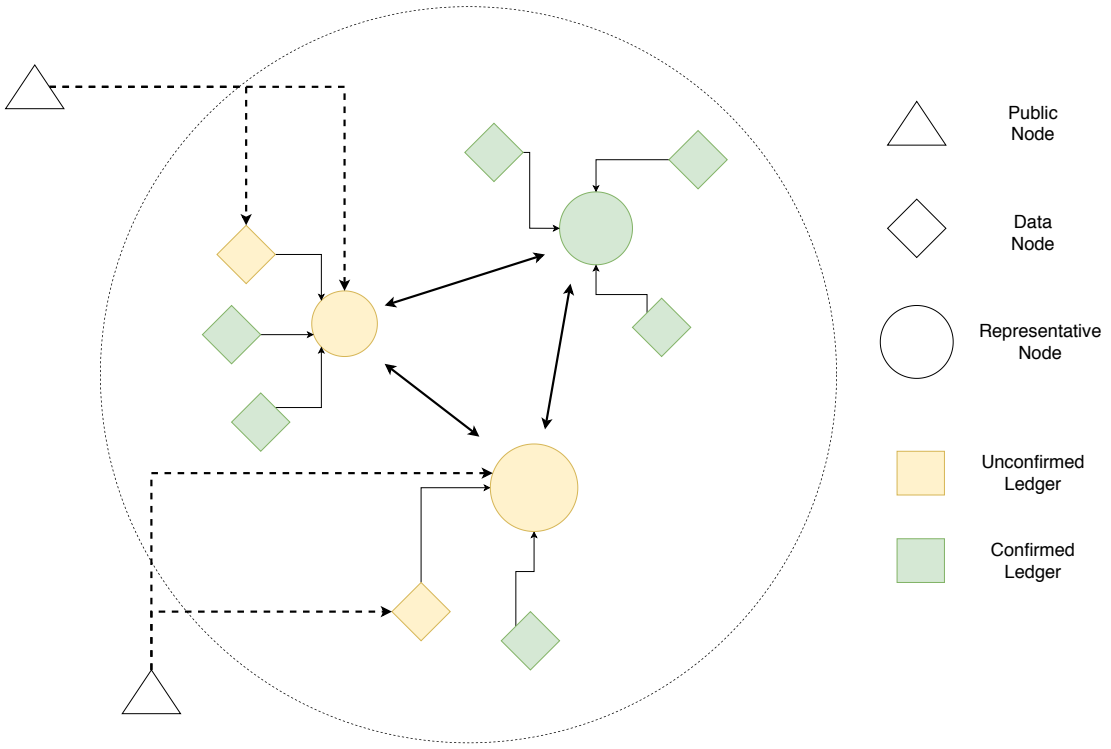

Figure 4-3: Verity Network Overview. Consensus occurs between *representative nodes*, with larger nodes indicating higher voting weight. Public nodes can contribute or validate freely to the main network.

4.1.3 Consensus Layer

The consensus layer handles transaction verification and confirmation synchronizations among data nodes in the network. For all distributed ledger technologies, the consensus layer is perhaps the most important, and an overview of consensus protocols and their comparisons for Verity's use case is described in 3. Before understanding how consensus fully works, it is important to first understand how Verity transactions are formed, verified, and processed in the network.

A **Verity transaction** can either occur between two *data nodes* (can occur within a data node) or a *public node* and a *data node*.

Insert Transactions

For transactions that occur between a public and data node, the transactions, as mentioned before, are *one-way* transactions from a public node to a data node, triggering

$$data\_node.insert(record)$$

.

The record that the public node sends to the data node must consist of the following data:

Listing 4.4: Public-Data Transaction

```
{
    "public_address": 256-bit integer,
    "type": insert,
    "record_metadata": {
        "metadata_json": ""
    },
    "timestamp": UNIX timestamp,
    "destination_address": 256-bit integer,
    "signature": 256-bit integer,
    "transaction_hash": 256-bit integer
}
```

The *public address* is the address which identifies the *sender* of the data and is retrieved from the sender's account entry. Thus, if a user were to send 5 records, it would originate from the same public address. The *record metadata* is the data that the user wishes to submit to the database, and will follow the constraints imposed by the administrator of the data node in the *application layer*. As expected, the destination address is the public address identifier of the data node. Finally, the

transaction hash is a hash of the original transaction body, that representatives use to ensure that no data has been altered by a data node.

In tandem with this transaction, the public node also broadcasts an additional record to the network, which is sent to a representative node. As mentioned in Section 4.1.1, these transactions are sent to a random representative, who is responsible for broadcasting the transaction to other representatives for a quorum described in Section 4.1.3. These transactions are similar to Public-Data transactions, except that there is an additional field describing the address of the destination representative node.

Listing 4.5: Public-Representative Transaction

```
{
    "public_address": 256-bit integer,
    "timestamp": UNIX timestamp,
    "transaction_hash": 256-bit integer,
    "representative_address": 256-bit integer,
    "type": representative
}
```

After this transaction has finished, the representative node recipient records the transaction in its list of *unappended transactions* and then, after receiving a handshake from a data node, will begin the quorum.

Deletion Transactions

The simplest transactions that can occur are transactions that occur within the same data node. These are only *data_node.delete*(). If a data node owner submits a record change via deletion, no quorum is necessary. All a data node must do is append a *delete* transaction to its ledger and its representative, who then broadcasts the updated data node's metadata (the latest transaction hash).

Listing 4.6: Deletion Transaction

```
{
```

```
    "insert_transaction_hash": 256-bit integer,
    "timestamp": UNIX timestamp,
    "signature": 256-bit integer,
    "transaction_hash": 256-bit integer,
    "representative_address": 256-bit integer
}
```

Here, *insert_transaction_hash* is just the transaction hash of the record that is to be deleted, and the *transaction_hash* identifies the outgoing transaction. If the inserted record has not been confirmed/verified in the data node's ledger yet, it's confirmation status will be set to stale as the record no longer exists in the db. The verification process is discussed next.

Verify Transactions

A verify transaction occurs between a validator type public node and a data node that has the *quality_verification* boolean set to *true* in its account. When a validator submits this transaction, it must submit the following data to a data node,

Listing 4.7: Verification Transaction

```
{
    "validator_address": 256-bit integer,
    "node_address": 256-bit integer,
    "verification_state": boolean,
    "proof-of-work": object,
    "transaction_hash": 256-bit integer,
    "record_transaction_hash": 256-bit integer,
    "timestamp": UNIX timestamp,
    "type": verification,
    "signature": 256-bit integer
}
```

All of the fields are self-explanatory save for the *verification_state* and *proof_of_work*

fields. When a validator validates or mines a record, it takes the following steps,

1. Requests a record to verify from network representatives. Representatives select a random eligible data node with >1 unconfirmed transactions. This is any unconfirmed insert transaction that is *not stale* (has not been deleted from the db).

2. After retrieving an unverified record from a data node, the validator verifies its quality according to the data node's specified verification task.

3. To avoid network spam, the validator additionally computes a small proof-of-work task designated by the network, and submits the entire transaction to the network.

Representative Node Consensus Quorum

A quorum consensus in Verity occurs the same way it would in the Nano block lattice system [17]. When a representative node receives an insert transaction from a public contributor, it requests the transaction hash from the data node recipient if it does not already have it and compares the hashes. If the hashes are equivalent, the transaction receives a **confirmation weight** which corresponds to the voting weight of the representative node. Once the transaction has received a voting weight greater than a threshold amount, the transaction is confirmed and can be verified by validators.

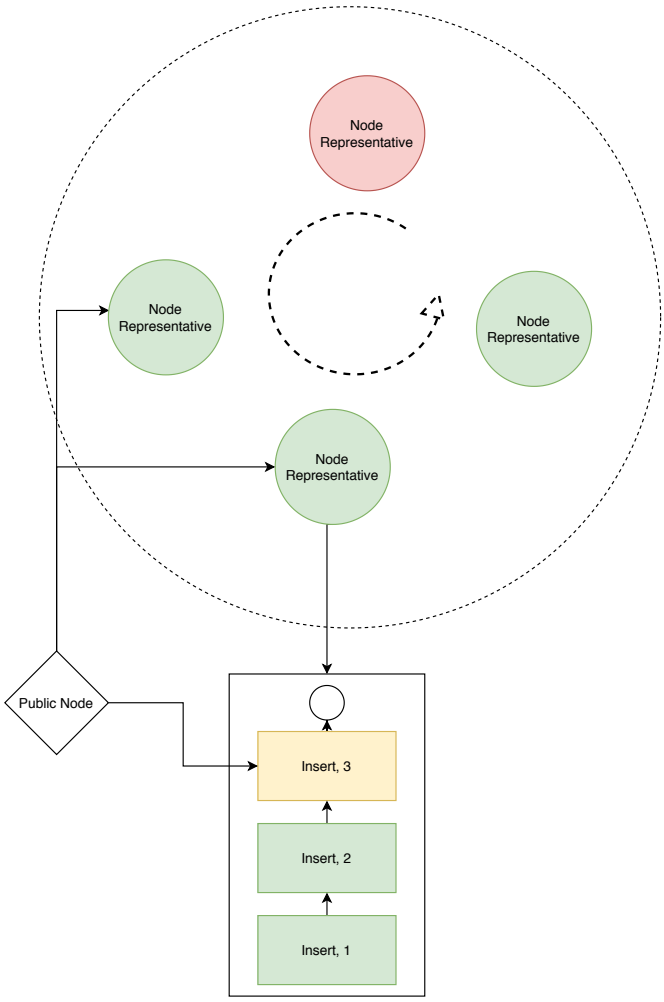

Figure 4-4: Representative Quorum. When a public node submits a transaction to the network, it also submits it to node representatives, who broadcast the network and vote on its authenticity. Here, three of four representatives ($> 50\%$) voted in favor of the authenticity of the transaction, which allowed the data node to append it to its ledger for confirmation from a validator.

4.1.4 Application Layer

The application layer describes how the nodes use their existing databases to work with Verity, and can be thought of as the user-facing interface. Because the focus of this thesis is on the Verity protocol/architecture rather than on its interface, only a brief overview of how an individual who wishes to use Verity would go through the

process is provided.

Data Nodes

For a database to become part of the Verity network, it would need to apply to the network by opening a Verity account. The *open*() function would then require some pre-configuration from the database admin to register itself in the network ledger to be sent to representatives. The account fields described in Section 4.1.1 describe the required parameters. One a database admin has successfully opened a Verity data node account with its database's header information, the data node will be tied to the dataset it represents upon access.

For the public accessing the dataset, it would be easy for them to check on the dataset quality. By building a simple ledger explorer, they could check the verification/quality status of each of the records and access records that meet a specific standard. This is main use case of Verity, which is to provide the public with authentic, high quality data.

Contributors and Validators

As mentioned in Section 4.1.3, contributors can directly participate in the network by simply adding data to datasets that use Verity and any Verity representative node.

Validators, on the other hand, would need to open an account with Verity using fields described in Section 4.1.1.

Chapter 5

Conclusion and Future Work

5.1 Future Work

5.1.1 Ledger Explorer (User Interface)

Although not the focus of this thesis, a block/ledger explorer can be built to provide the means through which a public data accessor can verify records [14]. A ledger explorer would simply *read* the Verity ledger and would have access to the state of all records by making use of the data stored in representative nodes and data nodes. As with most decentralized ledgers, the data stored is always public, and building an interface to easily access it is almost a requirement when building a new decentralized ledger platform.

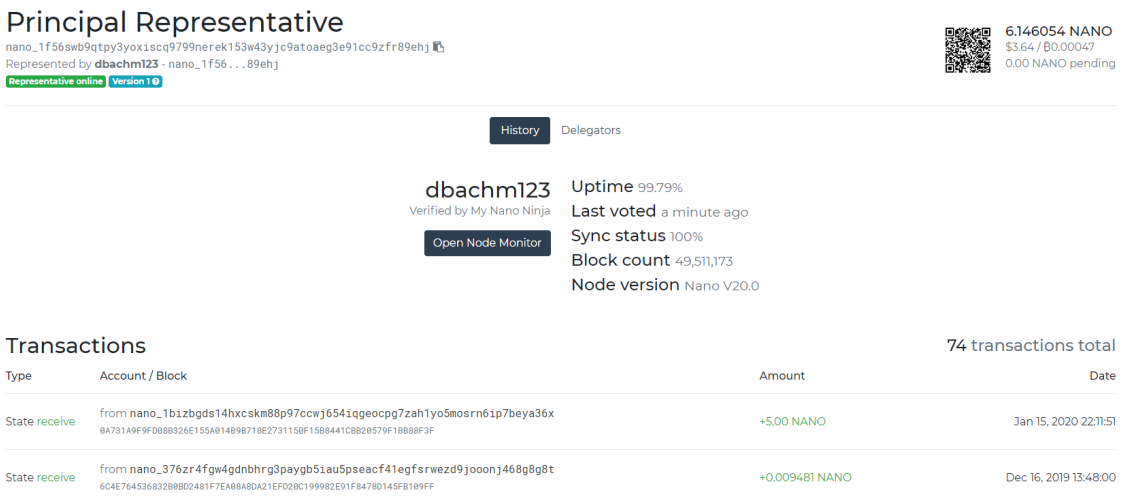

Figure 5-1: Nano Block Explorer Interface from nanocrawler.cc. Shows a Nano representative account with information such as balance, transactions, and voting weight

5.1.2 Validation Rewards

The original proposal for Verity focused on building a protocol that *incentivizes* contribution towards open datasets through blockchain based rewards. This, however, presents additional challenges that are out of the scope for this thesis. Namely, how to *mint* tokens, which refers to distribution, how to generate tokens, and the proper way to store them in Verity. This would require an additional understanding of the economics of token distribution in blockchain technologies [9].

Public validators in Verity currently have a *balance* field in their metadata which would allow the support of rewards, but requires further research. An existing protocol could be adopted such as those in Proof-of-Stake/Work systems like Ethereum or Bitcoin, simplifying the process. This would then allow for additional incentives to participate in the network as a validator.

5.2 Conclusion

The Verity Ledger presents a blockchain design to ensure authenticity, integrity, and validity while also promoting increased data quality in open datasets. Decentralized

systems are becoming more and more attractive as concerns for privacy and anonymity become much more prevalent. Whether it solves decentralized payments through solutions such as Nano or Bitcoin, creating a platform for smart contracts in Ethereum, or providing a ledger for microtransactions in IoT devices, distributed ledgers are solutions worth exploring.

Verity aims to solve the problems of data quality and authenticity in publicly-built datasets without comprimising the identity of any public contributor. Modern solutions include authentication schemes by requiring contributors to connect their own public accounts such as Google, Facebook, or Amazon credentials, which presents unnecessary hurdles for an individual. Because decentralized systems can be anonymous by design, this is no longer a requirement, providing an additional reason for contributors who do not want to forfeit their identity in order to contribute. In addition, those who access these data sets can be sure that the data they read is authentic, not tampered with by the administrator of the database, and in some cases, even manually checked for quality. With the benefits that Verity provides, an ecosystem of open, high-quality datasets can thrive, and with further improvements, could further support data built and created by the public.

Bibliography

[1] Bitcoin price, charts, market cap, and other metrics.

[2] Risks of blockchain projects- why so many blockchain projects fail?

[3] Martijn Bastiaan. Preventing the 51%-attack: a stochastic analysis of two phase proof of work in bitcoin. In *Availab le at http://referaat. cs. utwente. nl/conference/22/paper/7473/preventingthe-51-attack-a-stochasticanalysis-oftwo-phase-proof-of-work-in-bitcoin. pdf*, 2015.

[4] F. M. Benčić and I. Podnar Žarko. Distributed ledger technology: Blockchain compared to directed acyclic graph. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1569–1570, 2018.

[5] Erik W Black. Wikipedia and academic peer review: Wikipedia as a recognised medium for scholarly publication? *Online Information Review*, 32(1):73–88, 2008.

[6] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[7] Christian Cachin and Marko Vukolić. Blockchain consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*, 2017.

[8] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *Ieee Access*, 4:2292–2303, 2016.

[9] Sinclair Davidson, Primavera De Filippi, and Jason Potts. Economics of blockchain. *Available at SSRN 2744751*, 2016.

[10] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. Pbft vs proof-of-authority: applying the cap theorem to permissioned blockchain. 2018.

[11] John (JD) Douceur. The sybil attack. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, January 2002.

[12] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[13] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th {USENIX} symposium on networked systems design and implementation ({NSDI} 16)*, pages 45–59, 2016.

[14] George Foroglou and Anna-Lali Tsilidou. Further applications of the blockchain. In *12th student conference on managerial science and technology*, 2015.

[15] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[16] Seny Kamara. Proofs of storage: Theory, constructions and applications. In Traian Muntean, Dimitrios Poulakis, and Robert Rolland, editors, *Algebraic Informatics*, pages 7–8, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[17] Colin LeMahieu. Raiblocks: A feeless distributed cryptocurrency network. *URL https://raiblocks. net/media/RaiBlocks_ Whitepaper__ English. pdf*, 2017.

[18] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[19] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[20] Emanuel Palm, Olov Schelén, and Ulf Bodin. Selective blockchain transaction pruning and state derivability. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 31–40. IEEE, 2018.

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

[22] Serguei Popov, Hans Moog, Darcy Camargo, Angelo Capossele, Vassil Dimitrov, Alon Gal, Andrew Greve, Bartosz Kusmierz, Sebastian Mueller, Andreas Penzkofer, et al. The coordicide. 2020.

[23] Sarah Underwood. Blockchain beyond bitcoin. *Communications of the ACM*, 59(11):15–17, 2016.

[24] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International workshop on open problems in network security*, pages 112–125. Springer, 2015.

[25] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.