



VERITY COLLABORATION WHITEPAPER RELEASE CANDIDATE 1

A DECENTRALIZED GOVERNANCE AND REPUTATION
SYSTEM FOR BUILDING VERIFIABLE, TRUSTWORTHY AND
COLLABORATIVE COMMUNITIES

Matt Goldenberg, Eric Johnsohn, Doug King, Drew Blount

TABLE OF CONTENTS

Scope	5
Special Thanks.....	5
Abstract	6
Introduction	6
Ethereum	7
Summary of the Verity Platform	8
Reputation Layer	8
Governance Layer.....	9
Collaboration Layer	9
User and Application Layer	10
An Example	10
Reputation Layer	11
Reputation and Ratings Standard.....	12
Reputation Traits.....	12
Ratings and Attestations	12
Reputation Instances.....	13
Reputation Classes	13
Values-Based Reputation.....	13
Value Tags	15
Value Scores	15
Skills-Based Reputation	19
Skill Tags	20
Tokens for Skills.....	20

Contribution-Based Reputation	24
Utility Function Standard	25
Contribution Tokens.....	26
Verity Campaigns and Contests.....	26
Governance Layer.....	35
Governance Standard	36
Actions.....	36
Proposals	36
Roles	37
Permissions.....	37
Governance Systems	37
Value-Based Hierarchy	38
Contribution-Based Voting.....	38
Skill-Based Reputarchy	38
VerityDao.....	39
Making a Profit	39
VerityDAO Governance Actions	39
Collaboration Layer - objects in the verity ecosystem.....	41
Agents	41
Content.....	41
Communities.....	42
Creating a Community.....	42
Community Governance Actions.....	43
Users and Applications Layer	44

Example Applications	44
Decentralized Applications.....	44
Rating and Grading	45
Collaborative Crowdsourcing	45
Portable Reputation	45
Decision Making	45
Forecasting	46
Machine Learning Augmentation	46
Conclusion.....	46
Bibliography.....	47
Appendix A – Math Formulas and Algorithms	49
A.1 – EigenTrust Algorithm	49
A.2 – EigenTrust++ Algorithm.....	51
A.3 – Relative Rank Algorithm.....	53
A.4 – Affinity Score Algorithm	54
A.5 – Token Transfer Algorithm.....	54
A.6 - Token Transformation Algorithm	56
Appendix B – Mathematical Building Blocks in Contests.....	57
Utility Functions	57
Scoring Rules and Divergence Measures.....	57
Pooling and Extremizing Algorithm	58
Monte-Carlo Simulation	58
Appendix C – Technical Details and Challenges	59
Full Page Verity Architecture Flowchart.....	59
Expensive Computations	60

State Channels.....	60
Interactive Verification.....	60
Gas Costs	61
Short Term – Market Selection and Foundation Subsidies	61
Long-Term – Gas Automation and Website Subsidies	62
Randomness	62

SCOPE

This document represents Verity's technical approach to universal reputation, governance and collaboration. The intent is to show an approach that emphasizes a building block like structure that allows for flexible, extensible pieces that can be snapped together in powerful ways that truly solve some of the long standing problems with reputation and collaboration in both decentralized and centralized contexts.

This document is not intended to show Verity's roadmap, go-to-market strategy, competitive analysis, or market opportunity. For that, please see our **upcoming open source business plan**, which will go deeper into these issues.

This document is also not intended to show Verity's approach to personalized, customizable, and swappable reputation systems. For that, please see our **upcoming personalized reputation whitepaper**, which will go deeper into these issues.

SPECIAL THANKS

Special thanks to:

Matt Chwierut, Thai Wood, Rob Caraway, George Li, David Weiner, Jonas Goldenberg, Karen Weiner, Peter Borah and Kate Settle. This paper would not have been possible without you.

VERITY COLLABORATION WHITE PAPER

A DECENTRALIZED GOVERNANCE AND REPUTATION SYSTEM FOR BUILDING VERIFIABLE, TRUSTWORTHY AND COLLABORATIVE COMMUNITIES

ABSTRACT

We present a programmatic building block approach to enable social applications which depend on verifiable, trustworthy and collaborative communities. By layering reputation, governance and collaboration building blocks together, we show how our platform can be used in a variety of situations to model most community dynamics and cooperative goal oriented activities.

Throughout these layers, we present new reputation and governance mechanisms that express community characteristics in terms of agents' values, skills, and contributions. These reputation and governance metrics target and solve many of the underlying problems that have plagued internet communities in the past, and can be shared and transferred between communities, allowing for universal reputation metrics. These universal metrics can also incentivize cooperation and competition in a crowdsourcing setting, creating a task marketplace and rewarding those who offer the most value.

By putting these powerful tools in the hands of a user-owned decentralized autonomous organization, the Verity platform aims to fundamentally redefine trust and collaboration on the internet. By using an open reputation standard that can be used freely by applications and websites, Verity will create an ecosystem of interoperable trust and reputation. These tools simultaneously empower users, allowing community values and goals to be quantified and negotiated according to peer-to-peer trust. With dynamically-computed, peer-to-peer community values and reputation, social trust is brought to the decentralized web.

INTRODUCTION

One major feature of the internet is the emergence of myriad “online communities”: groups of like-minded people who find each other on some platform and build a shared identity. From usenet, listservs, and IRC, to Facebook and Reddit, online communities have always played a central part in the peer-to-peer economy and way of life of the internet. Yet while the internet is often considered a democratic and decentralized ecosystem, and users of social networks like to think of themselves as participants in a communal exchange of ideas, online platforms themselves are run in the interests of the companies that own them rather than the community members. Users can tell that platforms aren't working for them: they see their own information simultaneously withheld from their

personal use, and sold to private third parties. Because of this disjoint in information ownership, users get "locked in" to platforms, a state of things clearly against their own best interest. Meanwhile, content proliferates according to the platform's profit motive rather than the communities' values.

Verity solves these issues facing online communities by creating a reputation and governance system that is **decentralized** and **universal**. The decentralized aspect of the system means that reputation is computed according to a peer-to-peer algorithm that prevents centralized entities from controlling the trust network, and further, users govern themselves according to this reputation. The system is universal because it allows for reputation to be intelligently transferred between groups: because their members' values and reputation are quantified, the similarity between two communities is also quantifiable, and this defines the transitivity of reputation between two groups. This prevents any single group from locking in its members, while giving an informed basis for groups to judge a new members' reputation.

Verity creates standardized, decentralized, reputation and trust using dynamic peer-to-peer trust graphs. Communities can define their own values and types of reputation, yet reputation can always be transferred between similar groups. With quantified, peer-to-peer values and reputation, groups can be governed according to their own self-interest.

ETHEREUM

In late 2015, Ethereum launched, a platform for creating smart contracts backed by a blockchain. For the first time, decentralized applications (dapps) became almost as easy to build as standard web applications, and money was almost as easy to program as any other piece of data. By enabling arbitrary voting systems and financial contracts, Ethereum as a platform presents the possibility of dynamic, peer-to-peer, asset-holding communities.

While Ethereum represents a huge paradigm shift in the power to create applications that don't have a centralized failure point, many have pointed out that there are still fundamental limitations to the type of decentralization that can be achieved (Sztorc, 2016). In particular, critics have noted that any connection to external data feeds, and any logic that requires human judgment, cannot be decentralized purely with smart contracts. Verity fills this hole in the dapp ecosystem, bringing verifiable reputation and peer-defined values to any community. Ethereum smart contracts can now rely on trustworthy human inputs using this decentralized reputation system.

SUMMARY OF THE VERITY PLATFORM

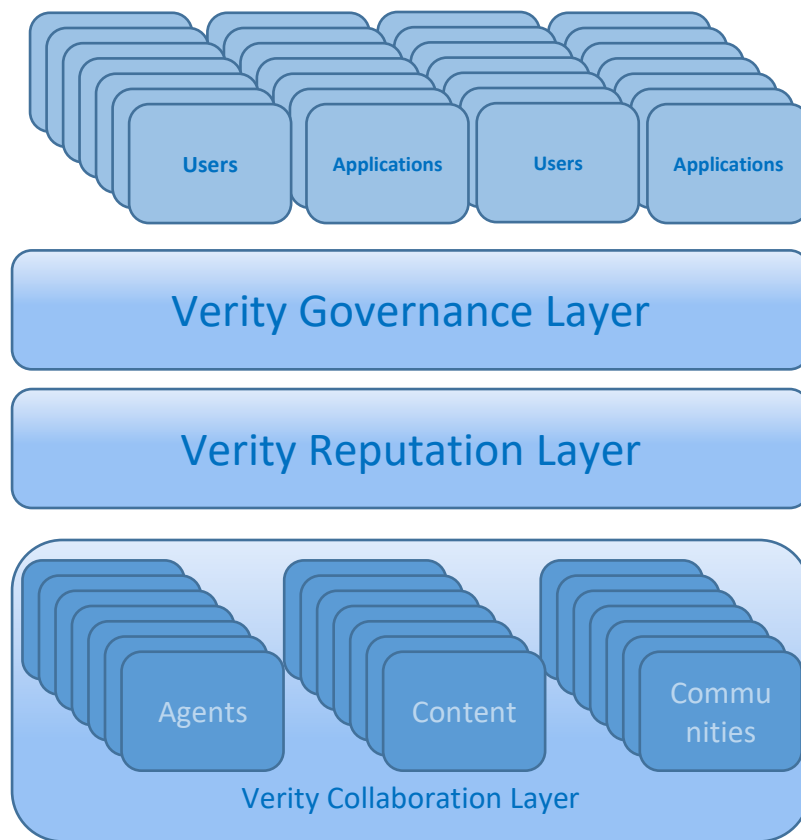


Figure 1 - Verity Platform

The Verity platform is a set of open source building blocks for collaborative communities and reputation-based applications. There are three layers of the platform: the **collaboration** layer that defines the makeup of a community, its members and values; the **reputation** layer that provides the tools for computing trust within and between communities, and the **governance** layer describing how communities can self-govern according to members' values, skills, or contributions. Developers building collaborative applications will use these layers as APIs that seamlessly integrate with smart contracts as well as traditional internet technologies.

This means that the Verity system serves as a sort of “reputation and governance” glue for collaborative applications, giving them a common language, the ability to share users and information, and the power to leverage common reputation systems and governance mechanisms.

REPUTATION LAYER

Verity's reputation layer consists of a general **reputation standard** for defining and computing peer-to-peer reputation metrics for both users and content. This allows for an ecosystem of plug-and-play reputation types to be developed which can be integrated into any Verity community.

On top of this standard, Verity builds its three foundational **reputation metrics** that can rate agents on the values they exhibit, the skills they possess, and the contributions they make to their community.

Values are ranked through a web-of-trust system, and are assumed to be transitive meaning those who exhibit values are assumed to be better at judging those same values in others.

Skills are ranked by a system of competition where agents who perform well earn reputation from those who perform poorly. In this way, skills based-reputation measures the relative skill-set of an agent compared to other agents.

Finally, **contributions** are ranked by measuring every agent's contribution to a pre-defined community goal. In this way, contribution scores are a measure of how much an agent has helped a specific community.

Verity is unique in its ability for agents to transfer reputation between communities in a way sensitive to those communities' similarity. This is what allows Verity to be a global reputation standard instead of another stand-alone metric such as Reddit karma or STEEM power. And while those platforms face difficult choices and friction between administrators and users when it comes to content and membership moderation, Verity's reputation layer quantifies social groups' natural, decentralized self-regulation: unwanted users and content lose reputation in their own group, and users cannot carry reputation between dissimilar groups.

GOVERNANCE LAYER

Verity's governance layer allows communities to make decisions and take action based on those communities' own governance rules. It implements a **governance standard** that allows for a flexible system of roles, permissions, and proposals. By layering these systems together with the power of smart-contract code, an ecosystem of plug-and-play governance tools can flourish.

On top of this governance standard, Verity introduces three basic forms of governance based on the three fundamental reputation metrics. The first of these governance types, **value-based hierarchy**, ties organizational roles to an agent's values-based reputation in the community. The second, **contribution-based voting**, allows agents to have political power proportional to the contributions they have given to the community, as measured by their contribution-based reputation. The third governance system is **skill-based reputarchy**, which gives agents influence commensurate to their ability to make good decisions, as measured by skill-based reputation.

COLLABORATION LAYER

Verity reputation is defined in the context of agents, content and communities. **Agents** are any users, groups, or autonomous actors that can perform actions within the system. A **community** in Verity is a group of agents that share a common purpose, vision, or goal. **Content** is anything in the Verity protocol that can be rated or acted upon, but which cannot itself act.

The purpose of the Verity collaboration layer is not just to define these objects, but also the social information necessary to define and preserve a community's group interest. There are two specific goals: to preserve community values over time, thus maintaining the identity of the community, and to incentivize actions in alignment with those values.

Checks and balances in the Verity protocol encourage community collaboration consistent with agreed-upon governance goals. Community core values and goals can be explicitly defined as a codified string of language, but the norms for what those values and goals mean in practice are flexible and quantified in practice purely according to user behavior and peer-to-peer rating. The Verity reputation system provides reputation values for community members for each "dimension" of community values defined in this way, and using the governance layer communities can be made to govern themselves according to these values.

USER AND APPLICATION LAYER

The **application layer** in Verity refers to the ecosystem of mobile apps, web apps, and decentralized apps that implement service calls to Verity's protocols. Verity's open source standards ensure that every innovation built on the platform is usable by all applications, and Verity's Ethereum backbone provides a common, decentralized data store for reputation. This allows reputation information to be shared and transferred between applications, and controlled by the holders of the reputation themselves. This gives Verity's application layer a unique network effect which can grow through any individual application gaining more users, or through more applications coming into the system.

Users and applications interact as peers in Verity's ecosystem, while using the protocol independently to meet their own diverse needs regarding identity, reputation, privacy, and access. Wildly different communities and business models alike can benefit from a shared language of reputation.

Verity's initial governance and reputation systems provide a variety of tools for user-owned projects, such as **contribution-based voting** and **skills-based reputarchy**. These solutions are designed to be attractive to users by sharing profit and putting choices in the hands of users. At the same time, they serve the needs of the application creator by reaching near ideal outcomes and incentivizing user participation.

This allows users and applications owners to work together cooperatively with aligned incentives and goals to deploy governance settings that best meet their joint needs. This is in contrast to user and application creator relationships of the past that put the needs of users and application owner at odds, such as selling user data or putting advertising on the platform.

Since the Verity protocol translates everyday user-application behavior into verifiable reputation metrics, the burden to manage "sensitive" metrics data is offloaded to the protocol itself. This allows applications to focus on core functions related to their specific purpose, while having access to metrics data when they need it.

AN EXAMPLE

As a concrete example of how Verity could be used, let's take the problem of finding agents who can write and audit smart contract code. In the Ethereum ecosystem, the hack of The DAO exposed this as an unsolved problem.

Using Verity, one would create a “smart contract security” community. This community’s reputation metrics, captured via Verity, would be shared by decentralized applications (dapps) needed in the ecosystem such as:

- A dapp for hiring qualified people to code your custom smart contract.
- A dapp for paid, crowdsourced smart contract security audits from qualified professionals
- A dapp for decentralized reputation-based commit access to open source smart contract projects
- A dapp to get advice from other smart contract coders on best practices (with the best advice from the most knowledgeable coders being ranked the highest)

Because each of these dapps would share the same reputation metrics and governance tools, each one would in turn improve the accuracy and utility of the other dapps, and the ecosystem as a whole would benefit from new dapps built on the platform in a way not seen with siloed reputation metrics.

Having gone through a high level overview of the system, the rest of the whitepaper will focus on the individual layers of the system.

REPUTATION LAYER

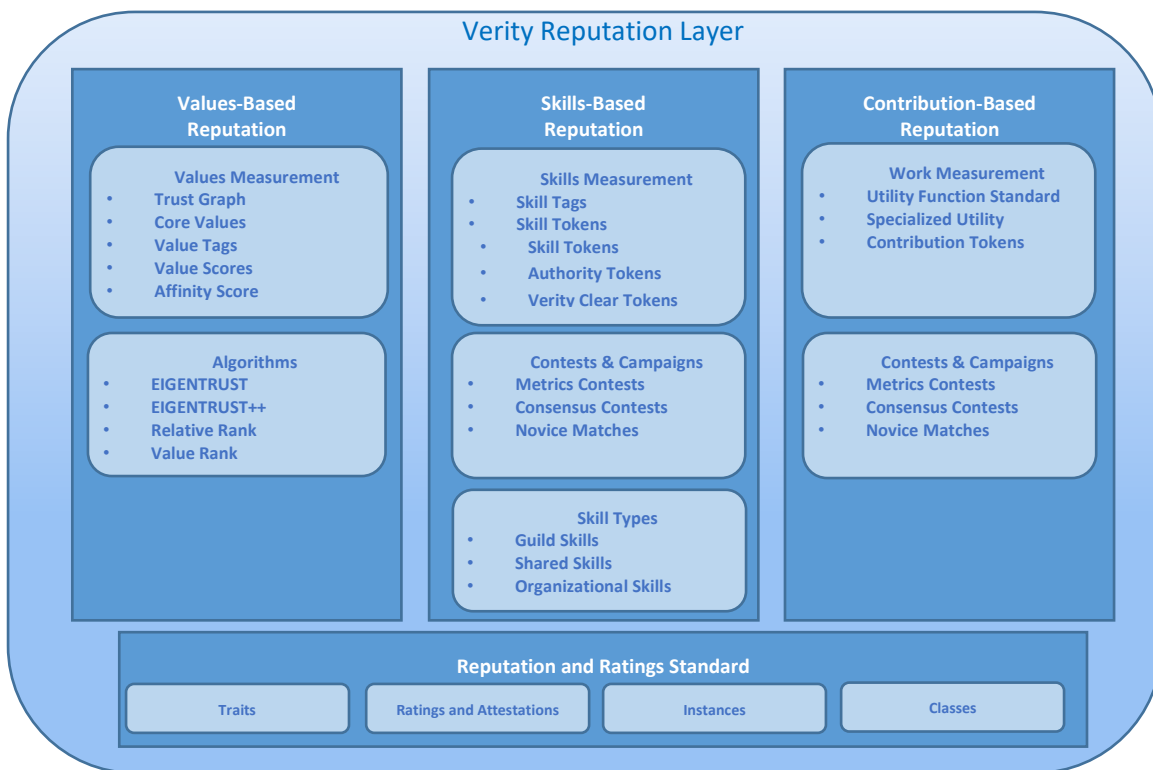


Figure 2 – Reputation Layer

Verity’s **reputation layer** provides a general purpose **reputation and ratings standard**. This standard acts as a simple interface for accessing and understanding reputation and ratings, similar to the ERC20 token standard (Vogelsteller, 2016). On top of this standard, we introduce three **reputation metrics** that represent the three types

of reputation typically used in real life scenarios. One is for **values-based reputation**, where an agent's values are rated based on others' opinions. The second is **skills-based reputation**. This is where an agent's skills are ranked based on its output quality, such as the clarity of a forum post or the performance of their code. Finally, we measure **contribution-based reputation**, which measures an agent's contributions to a community or organization over time.

REPUTATION AND RATINGS STANDARD

Verity's **reputation and ratings standard** is a set of common functions that allow smart contracts to implement ratings and reputation metrics in a consistent way, thus allowing them to keep the same interface for their smart contract even if they change reputation metrics.

The standard is simple and is built on top of Verity's notion of communities. It defines several functions to return information about the reputation or rating metric such as what it should be called, the type of reputation it is (categorical or numerical), and how it should be displayed (limits, units, etc.).

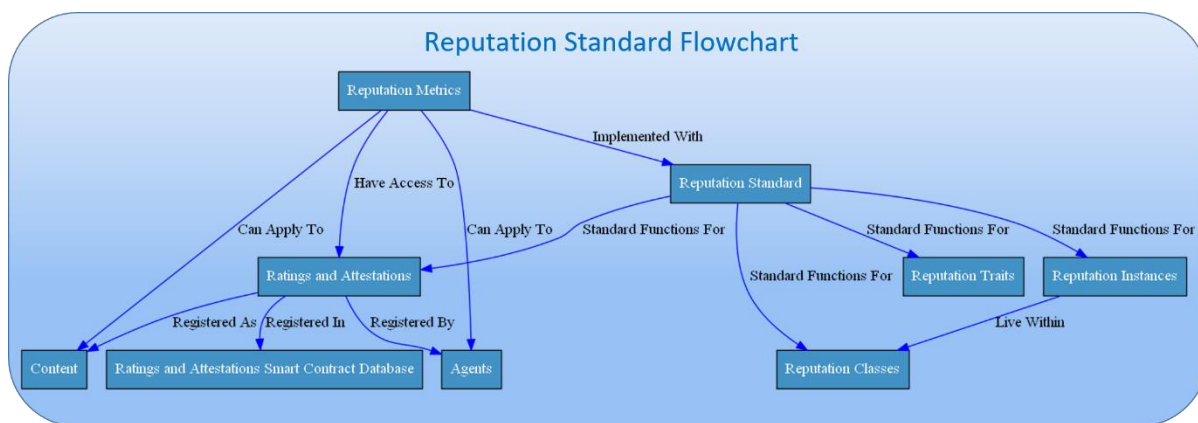


Figure 3 - Reputation Standard Flowchart

REPUTATION TRAITS

Every reputation type has **traits** that define the use cases for that metric.

- A reputation type can be **numerical** (meaning people are ranked in that reputation according to some number) or **categorical** (meaning that people are put into discrete buckets).
- Numerical reputation has an optional **minimum** and **maximum** trait.
- Reputation can also be used to rank **content** (such as ranking restaurants), **agents** (such as deciding how trustworthy someone is), or both.

RATINGS AND ATTESTATIONS

Many reputation metrics require agents to attest to or rate either content, or other agents. Verity's rating standard has a set of common functions to allow agents of a community to create **attestations and ratings**.

Crucially, an implementation of the ratings standard is separate from an implementation of the reputation standard. This allows different reputation types to be built on the same base data.

REPUTATION INSTANCES

Every reputation metric has an optional instance system, which allows the same reputation system, in the same community with the same traits, to be used to measure different metrics. For instance, a community could have one instance of "karma" that measures regular users and another that measures moderators. This system allows a community to use the same mathematical reputation rules to measure a variety of different things and allows new measurements using the same reputation rules to be created at will. It also allows communities to share the same instances, so that for instance a moderator in on one forum could carry their reputation with them and be known as a decent moderator on the other forum as well.

REPUTATION CLASSES

Our reputation standard is set up such that one can create different classes of the same type of reputation. These reputation classes all share the same basic building blocks, but apply to different communities, may have different traits, and do not share the same instances. This system allows for two communities to use the same *type* of reputation without sharing reputation between them. For instance, two separate communities could have a basic "karma" system with the same reputation type, but they would not have to share the reputation, or even have the same minimum and maximum for that karma.

VALUES-BASED REPUTATION

Existing ratings and reputation systems built into communities, such as Reddit, can classify the type of content that the community likes, but cannot identify the type of people that should hold influence in their community per community values. As such, any open community that uses such a content-centric system will be driven by populist values and conflicts. This causes communities to be held hostage by hostile dialog from outsiders, with centralized censorship the only recourse.

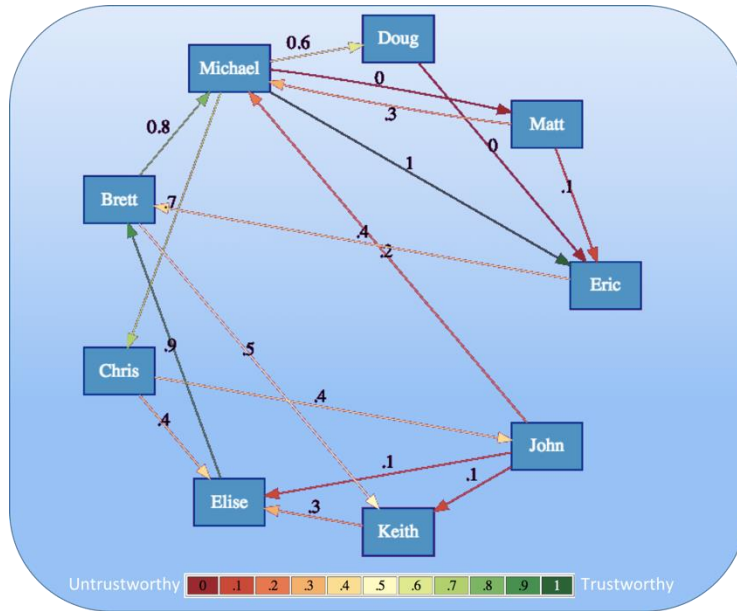


Figure 4 - Example Trust Graph

Verity's **values-based reputation** measures human values by analyzing a **trust graph** (shown above). In contrast to content-centric reputation metrics, it maintains community integrity without needing a centralized moderator. Using values-based reputation, communities can choose **core values** that they want their members to hold. Core values can influence various aspects of community involvement, such as ratings or stake in community decisions.

Each agent in the system is ranked with an **affinity score**, which shows how aligned they are with a community's values. Agents can gain influence in the community if they are in alignment with the values of that community. In this way, values-based reputation eliminates trolling and provocative behavior and enables moral decision-making, such as deciding what content should not be allowed on a platform.

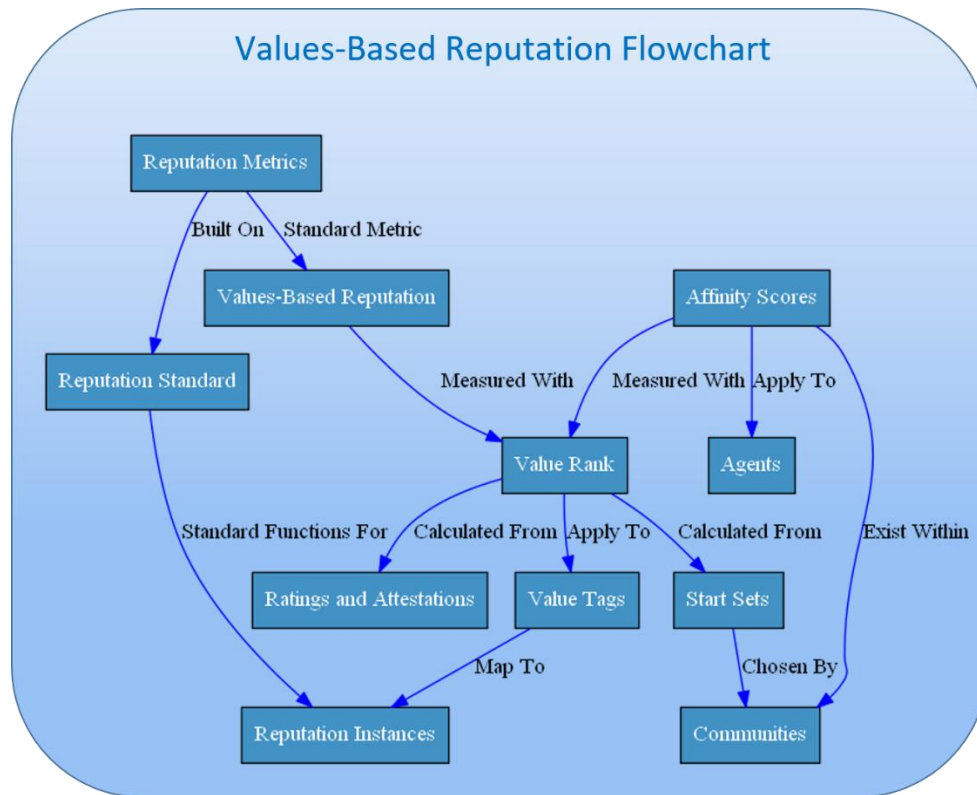


Figure 5 - Values-Based Reputation Flowchart

VALUE TAGS

Value tags are simple text representations of the value that will be used in a values-based reputation rating. Each tag has a “maximum” and “minimum” description that describes the value and its reverse. For instance, the honesty tag may have the maximum of “honest” and the minimum of “dishonest.” Each tag will be registered in a global registry along with a link to a longer free-form description of the value. Long form descriptions can be fuzzy and general, or precise enough to be used in a contractual agreement. For the purpose of Verity reputation calculation, tags and free-form descriptions are arbitrary and only have meaning in the context of the reputation trust graph of the community. The Verity algorithms will work with any language, jargon, or concept.

Tags and descriptions can be created and registered at will, or existing tags in the global registry can be reused if they are a good fit for a community. While agents can rate other agents and give personalized local scores to any agent for any value, global value scores are calculated only for values which are tracked by some community or group of communities, and are interpreted through the lens of that community's understanding of the value.

VALUE SCORES

An agent can have a **value score** for every value that a community measures, denoting how much that agent embodies that value, in the opinion of the community. At any time the value is between -1 and 1. Value scores

above 0 can generally be seen as having that value, while value scores below 0 can be seen as the reverse (as defined in the value tag metadata).

Value scores represent specific qualities that agents of the system hold. They're computed by combining versions of **Relative Rank** (Traupman, 2007) and **EigenTrust++** (Fan, Liu, & Li, 2012), two Sybil-attack resistant versions of the **EigenTrust** (Kamvar, Schlosser, & Garcia-Molin, 2003) algorithm. They normalize scores based on the number of ratings each agent has been given, and incorporate the structure of feedback that agents get respectively.

EIGENTRUST

The **EigenTrust** algorithm is based on the notion of transitive trust: a peer will trust agents trusted by those agents it trusts (and so on). The EigenTrust algorithm calculates a trust value by taking all positive interactions, subtracting all negative interactions, and then propagating this trust transitively along all agents.

The authors of the EigenTrust paper note that for networks with a sufficiently large set of agents, the aggregated local trust vector always converges to the same global number, regardless of which agent its calculated from. This means that the trust vector represents a global notion of trust that the network places on any given agent.

The algorithm for calculating EigenTrust is shown in [A.1 – EigenTrust Algorithm](#).

START SETS

The authors also note that there are a few problems with the simple algorithm above. Firstly, malicious agents can create networks specifically designed to increase the score count between each other, by “trapping” the transitive trust in tightly knit web of fake trust links.

Secondly, if an agent has not rated any peers (or has rated all peers negatively) transitive trust will be undefined, making the algorithm impossible to compute.

They solve both these problems with the notion of a trusted “**start set**.” This start set represents a set of trustworthy peers that have not been compromised. To remove the chance of the algorithm being trapped in a Sybil compromised network, they make sure that every peer has at least some small amount of trust allocated towards the start set, such that on any given step the algorithm can exit the malicious network by returning to the start set.

If an agent has not rated any peer agents in a positive way, then that peer is treated as having implicitly given high ratings to all agents in the start set, thus defining transitive trust for all agents and solving the second issue above.

EIGENTRUST++

EigenTrust++ suggests three ways to increase resilience above classic EigenTrust. Firstly, it adds the concept of **feedback similarity** to its trust propagation. This neutralizes a class of attacks that work by acting honestly while rating dishonestly; this attack can increase an agent’s reputation relative to peers who both rate and act honestly. Secondly, it incorporates information about how many feedbacks a peer has received, allowing for peers with lots of negative feedback to be treated differently than peers with low amounts of feedback. Thirdly, it creates **linear**

thresholds for propagating trust, reducing the possibility that dishonest collectives can pass on reputation to honest agents and vice versa.

For the purposes of this paper, EigenTrust++'s feedback similarity rating and linear thresholds are utilized, but not its incorporation of feedback number. This is because a different normalization procedure based on feedback number, described below in [Relative Rank](#) is used.

The algorithm for EigenTrust++ is shown in [A.2 – EigenTrust++ Algorithm](#).

FEEDBACK SIMILARITY

EigenTrust++ notes that a peer can maliciously attack the network by always acting honestly when interacting with high reputation peers but interacting dishonestly in other situations. It solves this problem by creating a **feedback similarity** metric which allows honest agents to detect this type of behavior, and transfer less trust to agents that engage in it. It has an added bonus in Verity: it captures the notion of subjective values – an agent will trust other agents that see the value in the same way that it does.

LINEAR PROPAGATION THRESHOLDS

EigenTrust++ shows that it's possible for a malicious collective to build reputation through a tightly knit network of fake trust ratings, then impose the collective's views on the rest of the network. To combat this, EigenTrust++ creates **linear thresholds**, which stops trust from being propagated to a new agent when those thresholds are exceeded. While EigenTrust's start sets minimize the damage of malicious collectives, linear thresholds go a step further and penalize these collectives. EigenTrust++ adjusts this threshold stochastically, to strike a good balance between penalizing clearly negative agents, while allowing agents whom were unfairly given negative reputations to still participate in the system.

RELATIVE RANK

Relative Rank is an algorithm that adds additional Sybil-resistance to the EigenTrust algorithm, while at the same time making it more suitable for peer-to-peer markets. Relative Rank creates a clear decision procedure to determine if a peer should be trusted within an interaction, by transforming EigenTrust's arbitrarily high trust vectors into a normalized value between zero and one. The normalization procedure also includes negative feedback, in order to separate dishonest agents from agents whom have simply not been ranked. In order to create this procedure, Traupman first analyzed the behavior of EigenTrust in marketplaces, then tried to determine a clear threshold in the determination of whether an agent was trustworthy or untrustworthy. Verity uses the same procedure but applies it to EigenTrust++.

The algorithm for calculating Relative Rank is shown in [A.3 – Relative Rank Algorithm](#).

VALUE RANK

Value Rank, the algorithm for calculating the values that an agent holds, combines the anti-Sybil methods from EigenTrust++ with the normalization methods from Relative Rank. It also makes two minor changes to the original Relative Rank algorithm. It makes local ratings more granular and allows for multiple start sets.

GRANULAR LOCAL RATINGS

In the original Relative Rank algorithm, feedback is binary and interactions could be rated only positive (plus one) or negative (minus one). While this makes sense for the original implementation of EigenTrust, in which a peer either gave the correct data or did not, it does not allow for the nuance that comes with arbitrary values, such as deciding the level of kindness that an agent showed. In Relative Rank, this also means that there is less distinction between agent ratings. For this reason, agent ratings are given as a decimal value between positive one and negative one, allowing for more granularity in every interaction. By multiplying the final rank by two and subtracting one, we can put local and global ratings on the same scale; this way agents get a clear intuition for what a Value Rank score means—an agent rating of -0.3 means the same thing as a Value Rank score of -0.3 .

MULTIPLE START SETS

In the original Relative Rank, a single start set is used. A related algorithm in the same paper called RAW also allows for personalized start sets. In Value Rank, multiple start sets can be created, one for every community that measures that value. This expresses values as they are in the real world: relative interpretations within each community, while still striking a good balance with computational cost.

COMMUNITY LEVEL VALUE SCORES

The Value Rank algorithm is able to calculate different value scores based on the initial start set. This is ideal for creating a global trust graph while allowing agents to build value scores faster in their communities. The ultimate goal for value-based reputation in Verity is to build a global **trust graph** that spans a diverse network of communities and scales to planetary levels.

COMMUNITY CORE VALUES

In Verity, each community creates its own set of **core values** and defines community members it considers paragons of those values. These members become the initial start set from which the trust graph grows. As new members onboard and participate in Novice Matches, they are rated by the existing members thereby adding new agents to the trust graph.

For more on Novice Matches, see [Error! Reference source not found.](#)

AFFINITY SCORE

As agents get rated on a community's core values, Verity learns their affinity to that community. This affinity can be mathematically represented by averaging the agent's value score along all core values.

This is called an agent's **affinity score**. The affinity score is used throughout the Verity platform to make sure that agents who have influence in a community are aligned with the values of that community.

The algorithm for calculating affinity score is shown in [A.4 – Affinity Score Algorithm](#).

VALUE SCORES IN ACTION

Imagine how value scores might be used on a site like reddit, with many different communities that can interact with each other. Here's one such example.

Firstly, each community chooses a set of core values. A simple example is "decency," which the "existential jokes" and "politics" communities both adopt as a required core value for participation in their community.

Although both communities chose the same core value, each chooses its own start set of users they consider decent. These users, in turn, rate other users within the trust graph, and the combined trust graph from both communities pinpoints who each community considers decent. This acts as a filter, keeping users in sandboxed versions of the discussion forums until they prove their decency. In doing this, both communities avoid trolls without a moderation team.

Secondly, aside from its core values, the "existentialist jokes" community has a particular brand of humor that it wants preserved even as the community grows larger. This community weights its votes and new posts by its own "humor" value ranking, allowing everyone to participate while still preserving the type of humor that it values.

SKILLS-BASED REPUTATION

Skills-based reputation is the second foundational type of reputation. While values-based reputation answers the question "What type of person are you?" skills-based reputation answers the question "What are you good at?" Skill measurements are based on agents' abilities to use those skills to fulfill the goals of the people they're working for, as determined by trustworthy other agents in the community.

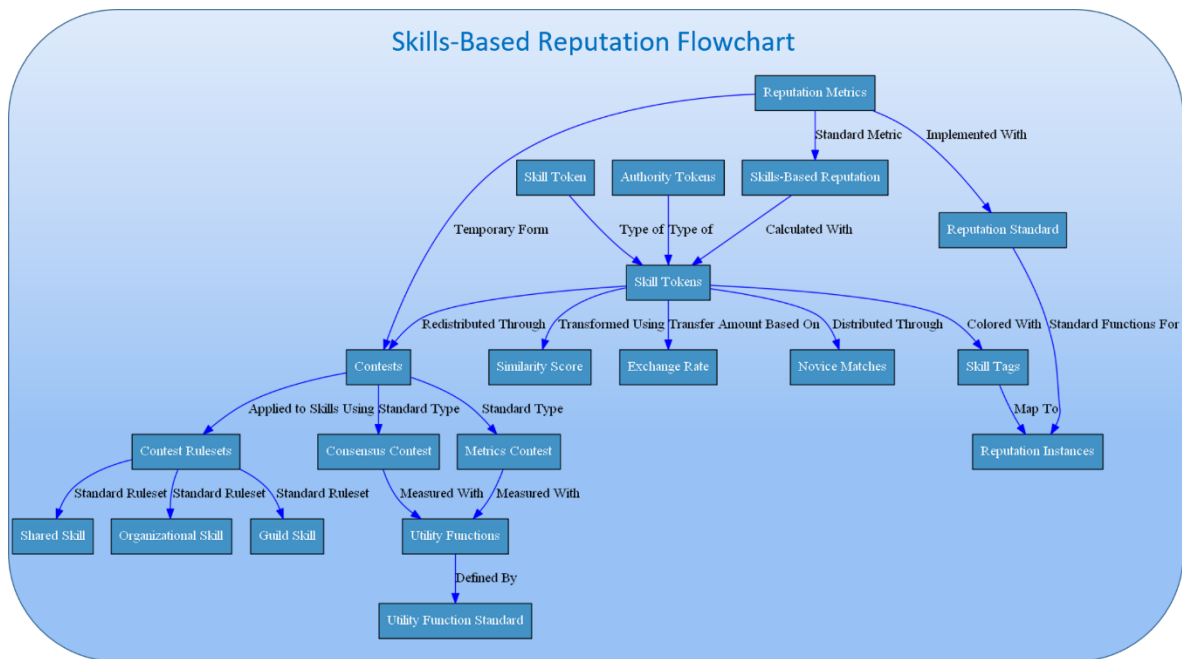


Figure 6 - Skills-Based Reputation Flowchart

SKILL TAGS

Skill tags are simple text representations of the different skills that agents can hold, along with a link to a longer skill description. They can be general and ambiguous, or specialized and precise. Skill tags can be created at will or existing skills can be reused if they fit the task at hand. Like value tags, the skill tags get their meaning not from the literal words in their descriptions, but from how they are used by communities.

While any agent can claim to have a skill tag, ratings for that tag only exist in the context of a community or group of communities. This ability to share skill ratings among groups of communities allows for the possibility of widely used skill measures that can become de facto standards.

TOKENS FOR SKILLS

Skills in Verity are implemented with three related types of tokens: skill tokens, authority tokens, and clear tokens. For any skill tag in a community, there are skill and judgment tokens, so we consider these two types to be akin to "colored tokens". Skill tokens measure, appropriately, a user's dynamically-computed degree of skill in the defined category. Authority tokens measure how reliably that user can rank others' skill, where "reliably" means "in a manner consistent with community norms".

Verity clear tokens that don't yet represent a specific skill in a specific community, and are used to initialize new skills tags in communities. Clear tokens are the only way that new skills are rated in Verity communities. Thus there are three types of skill tokens, representing three ways for users to contribute skill to their communities:

1. You can use the skill to solve problems for the community. This will earn you **skill tokens**.

2. You can grade agents' performance against some standard. This will earn you **authority tokens**.
3. You can buy tokens in a sale, or contribute to the VerityDAO. This will earn you **Verity clear tokens**.

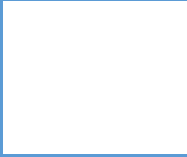
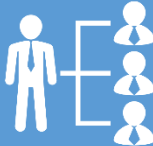

	Skill Tokens	Authority Tokens	Verity Clear Tokens
			
How to Earn	Earned for creating submissions that do well in a contest.	Earned for evaluating a submission accurately.	Can be earned by adding value to the Verity Platform.
How to Use	Can be used to elevate unranked submissions when entering a contest.	Can be used to weight evaluations when ranking a contest.	Can be bought and sold to provide "skill liquidity" and create new skill tags.

Figure 7 - Verity Token Types

SKILL TOKENS

To earn skill tokens one must generate new content. The more that a piece of content is valued by a community, the more skill tokens the creator earns. Submissions don't have to be in any particular form (though each submission has a string-based description), and community norms will define what constitutes a valuable submission. Thus skill tokens to be awarded for any tasks that the community feels comfortable verifying: programming, picking stocks, copywriting, videography, and others are obvious examples, but Verity communities are free to define and explore any other category as well.

AUTHORITY TOKENS

Authority tokens are earned when users rank content, and their rankings are later validated as being reliable in the context of the community, its values, and the contests within it (contests are described in [Verity Campaigns and Contests](#)). Specifically, your authority is computed as a function of how well you predict the performance of content in contests, weighted by the skills embodied in that content. So, if you make relatively accurate predictions about the success of new submissions with the skill tag "programming", you will be awarded "programming" authority tokens. Authority tokens then measure how much the community can rely on a user to make judgments related to a certain skill.

Since authority tokens are awarded as a function of making accurate predictions, they can be used to synthesize multiple users' predictions of the future into an authority-based, crowdsourced forecasting model. This type of crowdsourced forecasting combined with reputation has been shown to outcompete expert agents such as CIA analysts by 30% (Spiegel, 2016), and is comparable with the accuracy of prediction markets when the proper algorithms are used (Atanasov, et al., 2015). Thus, the inclusion of authority tokens alongside skill tokens enables prediction-related community functions similar to futarchy to be built with Verity reputation.

VERITY CLEAR TOKENS

Tokens representing a particular skill-tag, community, or token type are referred to as "Colored" Verity tokens in the tradition of "colored coins" where a given color represents a class of assets recorded on a blockchain. In contrast, Verity Clear tokens are Verity tokens that have not yet been marked for a specific skill or community in the Verity system. Verity Clear tokens are special in that they are not used as a measure of reputation in themselves. Instead, clear tokens provide "skill liquidity", allowing even skills that rapidly become popular to take an influx of new, skilled users, and allowing new skills to emerge. Verity Clear can be transformed into any of the tokens above, with any skill tag, in any community- but are limited by the agent's expected token score. This means that agents can only use Verity clear tokens to receive skills that the system predicts they would have earned anyway. This has the effect of allowing them to be sold and traded to anyone, but only used by people who have that skill.

PORTABLE SKILL TOKENS

One of the current problems with internet based reputation is that accounts can be easily sold if not tied to strong identities, thus making it impossible to know if reputation was earned or simply purchased.

Verity solves this problem by allowing the tokens used for reputation to be transformed into portable, sellable tokens, but distinguishing between tokens that are factored into reputation (activated tokens) and tokens which are not factored into reputation (deactivated tokens). The way it does this is by incentivizing the sale of or transformation of reputation tokens only to people who deserve those tokens, and deactivating the tokens if sold to others. By doing this it gives people honest ways to profit from reputation tokens they no longer have a use for. And while there is no way to prevent people from selling an entire account or identity in the verity system, tying reputation together across communities in a rich reputation graph gives people a strong incentive to stay attached to their current account and reputation. Meanwhile, the dynamic nature of the reputation graph ensures that a user's reputation reflects their current status in the community.

COMMUNITY SIMILARITY AND EXCHANGE RATES

To make skill-reputation portable, we measure the similarity between communities, and exchange tokens in proportion to those similarity measures. The similarity of two communities in Verity is found by assuming that similar communities will have members who hold similar tokens; communities are similar if people that have more tokens in community *A*, also have more tokens in community *B*. This relationship is easily calculated using the Pearson Correlation between all agents who hold tokens in each community. This score is then used to calculate the exchange rates of tokens between the communities, as will be described in further detail in the following sections.

The algorithm for calculating exchange rates is shown in [A.5 – Token Transfer Algorithm](#).

TOKEN DEACTIVATION

Token deactivation serves as a mechanism to make it hard for people to use their tokens to claim reputation in an expertise they don't have. Inactive tokens cannot be used in contests. The only way for tokens to get reactivated is through earning the same type of tokens in contests. For every token of the same type earned, an inactive token is reactivated. The flipside is that for every token of the same type lost within a contest, you lose a corresponding deactivated token. These lost deactivated tokens are then given back to the community to be distributed through Novice Matches. What this means is that owning deactivated tokens causes you to be able to both win and lose tokens twice as fast. This encourages agents to only be willing to hold deactivated tokens that they know they have expertise in.

When calculating how many tokens should be deactivated during the sale or transfer of a token, the agent's entire token portfolio, along with the exchange rate, is used to calculate an "expected tokens" value for the community they're moving into. Every token an agent holds in all communities (except the community for which tokens are being bought) is multiplied by its percent similarity with the community token being bought. These numbers are then averaged together, and weighted by affinity to the target community, to calculate the 'expected tokens' that the agent should have in that community. Any tokens purchased that exceed this number are deactivated.

This discourages whales in a community from buying or transforming tokens to cement their advantage, because they are likely already above their allotment of expected tokens. It also discourages non-agents from buying influence in contests, because their expected tokens are likely very low. What it encourages is agents from similar communities who don't want to sacrifice tokens through transformation to buy tokens from communities they should do well in. This transfer of crypto-reputation to people who deserve that reputation in real life is the exact behavior we'd like to encourage.

The algorithm for calculating Token Transfer is shown in [A.5 – Token Transfer Algorithm](#).

TRANSFORMING TOKENS USING SIMILARITY SCORE

When transforming tokens from one community to another (or one contest type to another), a slightly different procedure is used. The similarity between the two communities is used to figure out how many tokens are deactivated *in each transfer*.

This procedure holds up to the point at which the agent reaches their expected tokens for a community, adding up both their activated and deactivated tokens. After that point, the agent is no longer allowed to transform tokens into that community.

This procedure preserves the incentives of transferred tokens, and adds the additional incentive to transform tokens into tokens which represent similar skills. This allows skill tokens to take on a price related the value of that skill in the marketplace, instead of being merely an average of every skill in the market.

The algorithm for calculating Token Transformation is shown in [A.6 - Token Transformation Algorithm](#).

PORTABLE TOKENS IN ACTION

How might transferring and transforming tokens work in the real world? Let's imagine how this situation might play out with Carrie, a graphic designer who has earned her living for a number of years in a community centered around graphic design, and would like to expand her skills to the web design realm. Carrie could simply start from scratch building reputation in the web design community, but with Verity's skill-similarity-based reputation exchange rates, Carrie can leverage her skill in a related field to show her new community that she is reputable. Carrie could simply exchange her graphic design skill tokens into web design tokens, but she wants to stay active in the graphic design community and keep the reputation she has there. So instead, since Carrie can see herself that she'll be skilled in this new field, she decides to buy in with Verity Clear tokens.

To put this in figures, say that she has 500 tokens in the graphic design community, and her expected tokens in the web design community are 300 because the communities have a similarity score of 0.6. Carrie expects that it would take her 6 months to a year to gain the full 300 tokens in the web design community if she started from scratch. She does a quick estimate in her head and realizes that by having the skill tokens immediately she can make about \$10,000 more over the course of the year. Verity Clear tokens are selling for about \$20 on the open market, so Carrie decides that with her savings she can buy about 150 Verity Clear tokens for a total cost of about \$3,000.

Since she wants to start with 300 tokens in the web design community, she gets the remaining 150 by transferring from her graphic design tokens. Since the similarity score is 0.6, 90 of these 150 tokens are activated immediately, and the remaining 60 will be activated one by one as she earns her first 60 skill tokens in web design. By using portable tokens, Carrie can immediately get recognized and hired for better jobs when she switches industries, with reputation that is trustworthy, auditable, and based on her past performance in a similar skill. Carrie is immediately incentivized to contribute to this new community so she can restore her deactivated tokens---she essentially gains double skill tokens as long as she has deactivated tokens---and she recoups her investment in Verity Clear tokens by the second quarter simply by earning new tokens in the community. Thus her investment in web development was justified, and indeed paid for itself, because she had something to contribute to that community.

CONTRIBUTION-BASED REPUTATION

Contribution-based reputation is the third core type of reputation. While organizational skills-based reputation measures an agent's skill at adding value to an organization, contribution-based reputation measures how much value that person has added – that is, it takes into account the amount of work actually done. By using Verity contests to create clear measures for each action, a transparent, accurate method of ranking contributions emerges.

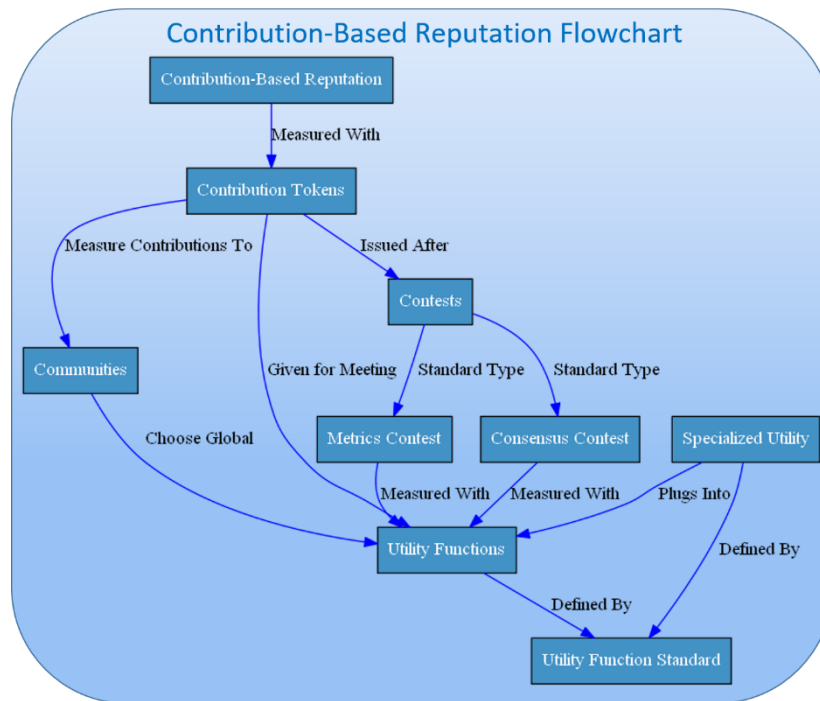


Figure 8 - Contribution-Based Reputation Flowchart

UTILITY FUNCTION STANDARD

To issue contribution-based reputation, a community must first choose a utility function.

A utility function is a function written in solidity that outputs a number. It indicates the community's preferences and goals – the higher the number, the more the community wants that outcome. Contribution tokens are handed out based on how much utility an agent has contributed to the community.

A community's utility function is chosen according to the governance mechanisms in that community, and can evolve over time as the communities' goals and preferences change, or the community realizes that their initial utility function didn't fully specify those goals and preferences.

All utility functions follow Verity's **utility function standard** – a standard framework that can be used by any governance mechanism that requires mathematical precision around goals (such as futarchy).

The utility function standard gives a serialized format that explains all input variables, a simple function to plug all relevant variables in that returns the utility, as well as standard functions to deal with specialized utility (explained below).

SPECIALIZED UTILITY

One common thing that communities want to do is measure specialized tasks, actions or roles. While a general utility function is needed at the community level, it may make sense to create customized utility functions that relate a community's goals to specific metrics or actions in an agent role. For instance, it may make sense to measure a phone support rep by customer satisfaction and calls per hour, and then have a standard way to relate these metrics to the broader community goals.

To handle this, Verity allows one to create **specialized utility functions** that relate to a specific task, and then plug those functions as inputs directly into the main community utility function. This process can be nested such that a specialized utility function plugs into another specialized utility function, and so on. This can be used to create agent metrics for specific roles, to give standard rewards for things like referring new members to the community, or even give a standard one-time reward for doing something like reading the employee manual.

CONTRIBUTION TOKENS

Contribution tokens are used to measure how well an agent helps a community meet its utility function. Contribution tokens work by assigning a value to actions that community members take, as measured by the utility function. Each community may issue its own contribution tokens, and allow contribution tokens to be transferred or forbid transfer and lock tokens into the agents who earn them.

CALCULATING CONTRIBUTION TOKENS

Contribution tokens are calculated by using Verity contests to measure the utility of a contribution. In the absence of contests, specialized utility can be used to give contribution tokens for certain actions as well. When a contest is complete, contribution tokens are minted and issued in accordance with how much utility a specific contribution gives to the community. Optionally some amount of contribution tokens can also be issued for accuracy, and critiques (using specialized utility) which are then distributed according to the rules of the contest.

VERITY CAMPAIGNS AND CONTESTS

Verity's skill and contribution tokens are earned and issued through *campaigns* and *contests*, which are the building blocks of community activity and self-expression. Using Verity's standardized set of building blocks and reputation API, as well as calls to arbitrary smart contracts, campaigns can augment arbitrary community activity with contribution and skill-based reputation. Basic examples of campaigns include producing news feeds or items of entertaining content, or developing a video according to certain specifications.

This section describes the Verity campaign and contest standard in detail, including the allocation of reputation tokens, and gives some examples of how the activities of familiar online communities fit into the Verity campaign framework. Note that campaigns are distinct from the Verity Governance protocol, which is defined in the next section, though both involve reputation in similar ways. Campaigns are the means of earning skill- and contribution-based reputation, and represent the regular activity, be it work, communication, or entertainment, of

communities; the Governance layer describes how community rules are changed by the communities themselves, including the rules of new campaigns and contests.

CAMPAIGNS

A **campaign** in Verity refers to a single unified event in which agents compete to deliver a multi-faceted solution for a Client. Every campaign is paired with a **utility function**, which defines a number of different inputs, and outputs a single number that measures the quality of each solution.

Each campaign can optionally be broken down into a number of **contests**, one for each input to the utility function. In this way, different inputs to the utility function can be measured by different agents with different rules, allowing for specialization within the campaign and allowing agents to work together to maximize utility for the client.

CAMPAIGNS IN ACTION

For instance, a reddit-like site that aims to show agents the best agent created content might have several criteria that it wants to consider when ranking posts for agents:

- An objective criterion, such as how many pageviews the post gets.
- The will of the agents, expressed by upvotes and downvotes.
- An agent's tastes, based on their past history of upvotes and downvotes.

Each expert can then compete in a variety of contests, depending on their expertise. For example,

- They can choose to create content that they believe will perform well along all metrics
- They can predict how many pageviews the post will get, and get tokens for being correct
- They can upvote or comment content that they think will be upvoted by others and get tokens for being early
- They can choose to recommend specific content they upvote to similar agents, and get tokens if those similar agents end up liking it.

In Verity, the only way to earn skills-based reputation tokens is by entering into a contest. A contest is a community level event in which an agent competes against other agents in the community to create the best content.

Contests can be repeating, such as trying to create the best content of the day in a certain subculture. They can also be one time, such as a company making a contest where agents compete to create a new product line. Verity's standard contest types all go through a standard process, during which agents compete with Skill tokens and Authority tokens and are awarded with contribution tokens.

CONTEST STANDARD

All contests are implemented as smart contracts that follow a predefined standard. The contest standard defines a temporary, individualized type of reputation that only exists for the duration of the contest. Using Verity's standard tokens, these contests can then be combined together into a single reputation type which can use different rules depending on the use case. This allows new crowdsourcing applications with varied needs to incorporate Verity reputation into their own unique contest types that are not covered by standard Verity contests. It also allows for various contest types to be experimented with, such as the algorithms provided by fellow crypto-reputation platforms such as Backfeed, Augur, Steemit, or Synereo.

CONTEST TYPES

There are two basic types of contests in Verity. With these two contests, many criteria can be included in a utility function. Those contest types are

1. **Metrics Contests:** Contests in which agents compete against some objective criteria provided by an oracle.
2. **Consensus Contests:** Contests in which agents compete to please or agree with their peers

We consider these to be somewhat fundamental contest types, but the Verity reputation API can be used to create custom contest types as well as described above. For more on the mathematical basis of these contests, please see [Appendix B – Mathematical Building Blocks in Contests](#).




	Metrics Contests	Consensus Contests	Custom Contests
			
Description	A contest that uses objective criteria to rank participants.	A contest that uses authority-weighted voting to rank participants.	A contest created using our contest standard.
Details	Each submission is ranked by a weighted scoring rule.	Each submission is ranked by a weighted divergence measure.	Standard functions describe novice matches and scoring rules.

Figure 9 – Verity Contest Types

METRICS CONTESTS

Metrics contests are contests in which agents' probabilities are ultimately judged based on real world data provided by an oracle (an agent that takes information from the real world and posts it to the blockchain). This oracle could be a data feed provided by a provider like Oraclize it, a Schelling-point oracle such as SchellingCoin

(Buterin, 2014), or any other compatible smart contract. Metrics contests can be straightforwardly implemented to function like crowdsourcing sites such as kaggle.com and gjoin.com, which both use objective real-world criteria and deterministic algorithms to rate their competitors in contests.

METRICS CONTESTS IN ACTION

How might metrics contests be used in the real world?

Imagine a “Conversion Optimization” community that had experts at taking a company’s website, and maximizing the amount of website viewers that converted into paying customers. By integrating this conversion optimization website directly into their existing conversion optimization software, the metric of “conversions” could be read directly from their website. Experts could then collaborate and compete to create versions of the website that they think would convert better, and forecast how each variant would convert. The top forecasted designs would then be shown to visitors of the company's website using A/B testing, and the forecasts would be compared to reality. Both payments and reputation would be distributed based on the actual revenue metrics that the company cares about.

CONSENSUS CONTESTS

Consensus contests measure community agreement on a subject. They can be used as Schelling-point oracles similar to Augur’s reputation metric, to rank options against each other based on their community acceptance as in Steemit, or to gauge a community's take on intangibles such as rating how beautiful a piece of artwork is. This is analogous to crowdsourcing sites such as the tech advice site StackExchange, which use consensus based mechanisms to rank their participants.

CONSENSUS CONTESTS IN ACTION

Firstly, one can imagine a StackExchange-like forum where programmers could discuss the creation of smart contracts. The entire site would be a daily consensus contest to ask the best question, and the questions themselves would be contests to determine the top answer. One would get Creativity tokens for asking and answering questions and Accuracy tokens for voting on them. Instead of a single up or down vote, voting would be more like range voting, in which you could allocate however many votes to every question or answer you voted on, and the entire distribution of your votes would be interpreted as a categorical distribution, with your number of votes indicating something akin to your probability estimate that this is the best question.

Secondly, as mentioned above, consensus contests could be used as a kind of decentralized oracle for metrics contests, using the game theoretic mechanism of Schelling Score. If you made a consensus contest for instance around how many bugs had been found in a contract over the past year, you’d be incentivized to count the obvious bugs that everyone else would also count, and everyone would most likely cluster around the same set of numbers at roughly the same probabilities. This allows you to have metrics contests without any worry of centralization.

EARNING NEW TOKENS THROUGH CONTESTS

In addition to whatever reward was posted into a contest by a client, agents also earn newly minted tokens by participating in Verity contests. These tokens are given proportionately to each contest based on how many agent tokens are in that contest, and then proportionately to each agent based on their score within that contest. These tokens are activated if the agent is below their expected tokens for the community, but are given to the agent deactivated otherwise. Agents can choose to sell these deactivated tokens on the open market, or to keep them in hopes that they will be able to activate them in future contests.

A portion of newly minted tokens (chosen through community governance) go to agents through agent contests, and the rest goes to novices through participation in Novice Matches (explained below).

ECONOMIC VALUE OF TOKENS

Where Verity tokens get their economic value is an important consideration for the platform. Verity tokens having a non-zero price on the open market is important for a number of reasons:

- Allowing agents to sell reputation legitimately, disincentivizes agents from illegitimate methods of profiting on reputation that can destroy the predictive power of it, such as selling their account.
- Requiring agents to buy into similar communities isolates each community from attacks on adjacent communities, creating a cost for every subsequent community the agent would like to attack.
- The price of reputation creates an incentive for agents to participate in communities that may be socially but not economically lucrative such as non-profits, by giving them the chance to sell their tokens to people whom are looking to prove those same skills in for-profit ventures.

This means that it's important to do some reasoning about the economic properties of reputation token, and where they may derive their value.

PRICING ACTIVATED TOKENS

An activated token is any token which an agent purchases while still below their expected tokens for the community they're purchasing from. The central reason that an agent would want to purchase activated tokens instead of earning them on their own is to be able to get more rewards faster from participating in contests.

Therefore, the price that an agent should be willing to pay for tokens is determined by the net present value of the rewards that an agent would get when buying the activated tokens directly minus the net present value of the rewards that an agent would get over that same period it would take them to earn those tokens through competing in contests. Thus the relevant factors for reputation price are:

- How long it takes to earn new reputation tokens
- How economically valued the reputation token is within contests.

This seems to match our intuitions about how reputation should be valued – Reputation that is harder to earn and more economically valuable tends to be seen as more valuable in society.

PRICING DEACTIVATED TOKENS

When looking to price deactivated tokens, similar considerations are taken into account. However, there are two additional caveats. Firstly, the amount of time that can be shaved off by buying deactivated tokens is significantly reduced because getting the benefit from them requires participating in contests. Secondly, there's a third factor besides the hardness and economic values of contests – there's the agents probability estimate of how many reputation tokens they'll earn and lose in that period.

For the purposes of clarity, make three simplifying assumptions. Firstly, it will take agents twice as much time to earn tokens the normal way as it will to earn tokens by buying them deactivated. Secondly, this means the agent will be able to earn double the rewards over the same time period that it would take them to earn those tokens the normal way. Thirdly, their ability to earn those tokens is a simple binary situation where the agent either gains all the deactivated tokens with some probability, or loses all the deactivated tokens.

In this case, the agent should be willing to pay double the net present value of earning the tokens the normal way, multiplied by their own confidence interval that they will indeed earn all those tokens. Once again, this matches with our intuition of how reputation should be bought and sold in the real world, such as in the case of an agent buying a brand in the assumption that they'll be able to maintain the same value as the previous owner.

PRICING VERITY CLEAR TOKENS

The price which an agent is willing to pay for Verity Clear tokens should track the price of the highest price activated token at any given time, as they can be turned into these tokens at no cost. However, this assumes that the value of buying token in an existing community will always be more than the value of creating a new community.

Any new website that wishes to use our technology in an entirely new community (and therefore get most of the early stake within that community) will want to purchase Verity Clear in order to seed that community with some agents. Thus the value that a community creator would be willing to pay is determined by the net present value of the expected future profits from that community (which they can get either by being an agent in that community, or a client.)

Thus whether or not Verity Clear tokens are being bought for community creation or reputation will fluctuate. As new use cases for Verity are created, it will be more attractive to create new communities to take advantage of those use cases. As the websites behind those communities begin to grow, it will be more attractive to transform the tokens into reputation tokens. Then, as the communities mature, this will in turn enable new use cases, which will again create the incentive to create new communities.

ONBOARDING AGENTS IN VERITY

Verity has two distinct ways to earn its tokens.

1. **Novice Matches**, involving contestants who are working to earn newly minted Verity colored tokens.
2. **Contests**, involving agents gaining and losing tokens from each other, in addition to earning newly minted and possibly deactivated tokens.

By separating the two ways to earn tokens, there is a smooth path for which agents can work towards earning money on the platform while learning community norms and improving their skills.

NOVICE MATCHES

Novice Matches allow new agents to grow their affinity score and earn Verity tokens as a new member of a community. They're scored very differently from agent contests, and must use a separate Sybil defense mechanism because they can't use token ownership as a proxy. Every contest contract can specify a corresponding Novice Match protocol, and if so, will have a mirrored version of all agent contests using that protocol. The general format of Novice Matches is as follows

1. Only accounts without Verity Tokens in that community can participate in the matches
2. All accounts must complete a nominal task such as a captcha to participate in matches for that month.
3. Only accounts with an affinity score of .5 can earn token in those matches.
4. Novice Matches are played out over a defined period. At the end of the period, all qualified participants are given a percentage of the newly minted Verity tokens, based on their performance in the Novice Matches that month.

SKILL TYPES AND CONTEST RULESETS

Any community can choose a skill tag that it chooses to measure using its reputation tokens, provided it has some Verity Clear tokens that it can seed its members with. The community simply chooses what **contest ruleset** it would like to allow to for its skill tag, and a variety of different behaviors emerge that can measure very different **skill types**. To start with, Verity will provide three sets of rules around contests, which allow three very different options for what skills mean. These three different rulesets allow a very flexible definition of skills that can encompass most use cases.

GUILD SKILLS

A **guild skill** in Verity allows any contest that has consistent values, but a customizable utility function. In practice, what this means is that the community defines what type of expertise it would like to represent, and then it allows any client to have its members fulfill that client's goals. The guild's challenge in this case is to define their values in such a way that they believe strikes a good balance in flexibility of clients vs. specialization of skills.

A real life example of the guild skill model would be 99designs.com. 99designs handpicks what it calls “Platinum Designers” – designers whom meet the criteria that 99designs values in designers (99designs, 2016). At the same time, 99designs allows for clients to choose their own criteria for their utility function, using what it calls “style attributes” to describe what features the client is looking for.

Instead of handpicking designers one by one, with Verity one could imagine a model where 99designs handpicked a few people it considered “good designers,” and use this as the start-set for the “good design” value. New platinum designers could then grow organically as members within the community rated each-other with that good design value.

SHARED SKILLS

A **shared skill** is a flexible skill type in which there’s some broad agreement around what goals a skill is trying to accomplish, but there many different values that different communities have that further refine that skill for their community. In practice, what this means is that the community defines a standard utility function for every contest, but allows each contest to specify its measurement for affinity, by which the skill tokens get weighted for that contest.

While there are currently no real life examples of shared skills, this is simply because no system like Verity currently exists. An example of where a shared skill might be useful is for a skill tag called “internet commenting.” A simple utility function could be defined, which would be a consensus contest where the goal was to up-vote comments that other members also up-voted. This same skill could be shared by Huffington post, The New York Times, Reddit, and YouTube.

In order to capture the different types of comments that might be considered valuable on different platforms, an agent’s tokens can then be weighted by affinity to each community’s own values. For instance, humor might be valued very highly in Reddit, but lower in The New York Times. Alternatively, YouTube and Reddit might both value humor, but have different ideas of what constitutes good humor. This mechanism captures these differences quite elegantly.

ORGANIZATIONAL SKILLS

Organizational skills are skills in which both the goal and the values are fixed. In practice this means that an organization has its own utility function, and people can prove their value to that organization by having values the organization values, and helping it maximize its utility. The tag will be something like ‘Adding Value to Verity’, and the amount of tokens an agent holds would represent how valuable they are to the Verity organization.

An example of how this works in the real world is pay scales. An organization tries to find people that fit in with its culture, then pays them based on how much value they add to the organization.

CONTEST COLLABORATION

While contests are a powerful tool, they suffer from a key deficiency – they encourage competition, but as a result discourage collaboration. To solve this, Verity introduces the notion of **shared credit**, and a particular type of

shared credit called sub-contests. These two mechanisms together allow for flexible collaboration in which people share in the rewards for a particular submission, critique, or prediction.

SHARED CREDIT

Verity's **shared credit** mechanism allows a smart contract to act as a proxy for any number of agents, whom each can contribute some of their skill tokens to the contract, without deactivating or otherwise jumping through the hoops that are normally required to transfer tokens. In the simplest example, a team of three people could work together to create a submission, prediction, or critique. They could use a shared credit smart contract that took the rewards due to that submission, and split them evenly three ways when they eventually decided to split their team up.

SUB-CONTESTS

Sub-contests are a particular type of shared credit rule that allow people to work off of other's creations. This allows the shared credit mechanism to represent situations like Wikipedia and git, where people are taking the best product and working to improve it, rather than competing to create the best product from scratch. The general idea behind sub contests is that each submission itself becomes a contest, in which derivatives of that submission are competing to be the best "version" of that submission. By averaging the percentage improvement in utility of each solution with the percentage similarity with each submission, one can figure out how much of the credit should go to the original solution, and how much should go to each of the sub-solutions.

By combining this mechanism with the simple shared credit mechanism, very complex behavior can be created, including behavior in which several solutions from sub-contests are combined into a single maximal solution, and skill tokens are distributed accordingly.

For example, imagine a group of editors collaboratively editing a chapter of a book. The chapter would be a submission, and a few sub-contests might emerge to edit various paragraphs into more readable versions. The best paragraphs from each content could then be combined, allowing for the best chapter overall to emerge.

GOVERNANCE LAYER

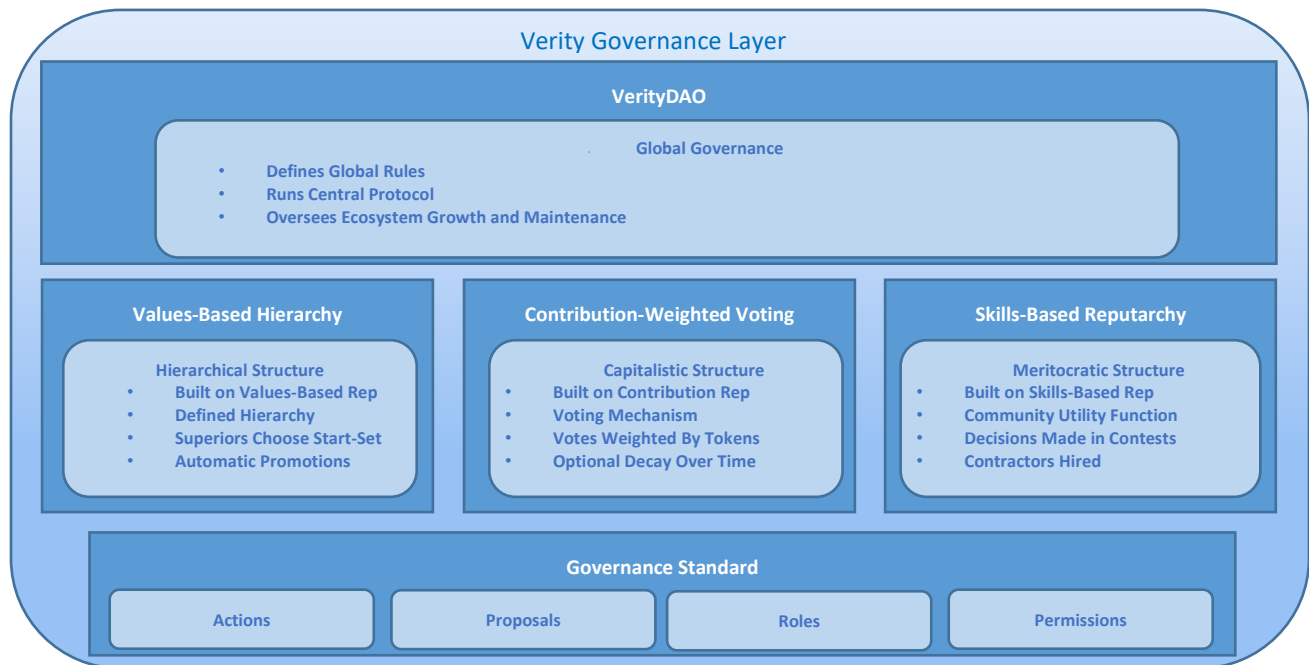


Figure 10 - Governance Layer

Verity's base governance layer at its core is a simple standard that can be extended with smart contracts to encompass arbitrary governance protocols. This standard defines governance for each Verity Community, as well as governance at the Verity protocol level itself. This top-level governance standard for Verity itself is called VerityDAO.

On top of this standard, we have built three basic forms of governance based on three types of reputation in the Verity system. The first is a hierarchical structure based on values-based reputation: a hierarchical organization that hold values, implemented as a reputarchy. The second, where votes are based on contribution tokens, is a contribution-weighted democracy. The third is a meritocratic structure using skills-based reputation.

In Verity, a community's governance rules are formalized at the point of creation. The plan for the VerityDAO (itself a Verity community) is to start as a hierarchical values-based reputarchy, in which the decisions are informed by the value-reputation of initial users such as the Verity Foundation, investors, and token holders, and their associated web of trust---but also to build in incentives similar in spirit to Ethereum's difficulty bomb¹ for VerityDAO to switch to both voting with skills-based reputarchy and paying dividends weighted by contribution as the platform matures.

¹ <http://ethereum.stackexchange.com/questions/3779/when-will-the-difficulty-bomb-make-mining-impossible>

GOVERNANCE STANDARD

Verity's governance standard is a simple, flexible standard that includes hierarchical roles, permissions, actions, and proposals. The systems also allow roles to be held by smart contract agents, which themselves can follow the governance standard in an arbitrarily nested fashion. By layering these simple building blocks together, complex governance mechanisms can be built, and mechanisms like Futarchy, Liquid Democracy, or Quadratic Voting can be easily swapped in and out as desired. Nexusdev's dappsys framework² already includes much of this functionality, and we're currently investigating the possibility of adapting it for our purposes.

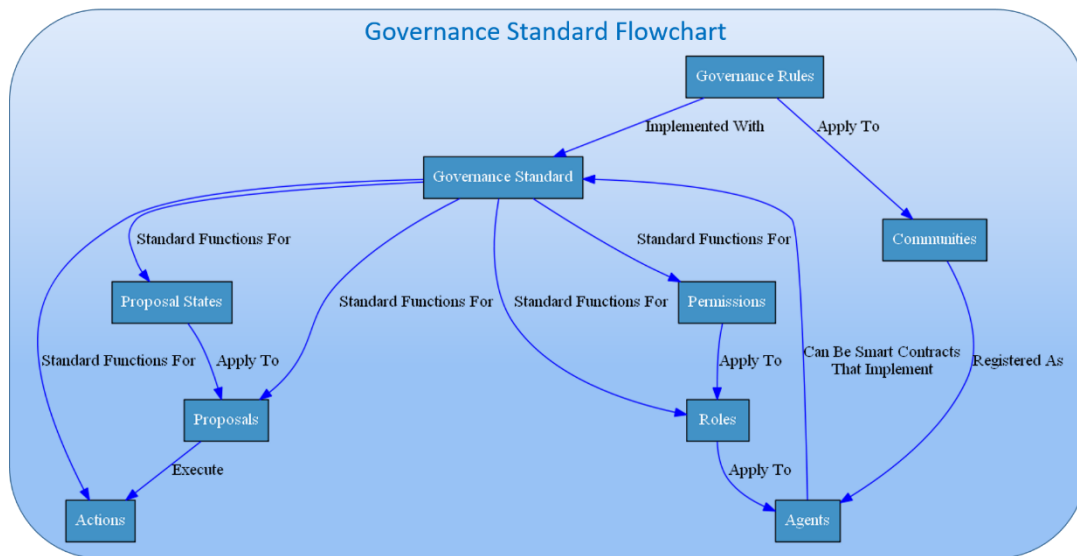


Figure 11 - Governance Standard Flowchart

ACTIONS

An **action** as defined by Verity is a specific function that lives within a smart contract and can be executed on the Ethereum network. The action could be a simple action that merely triggers an event that shows that the action happened, or can be a complicated action that involves invoking outside contracts or moving ether. The action may take arguments, which are inputs to the action that change its meaning, or they may simply be actions that have one meaning, such as "resign."

An example of an action in the US government might be "approve citizenship" which takes as an argument the identity of a potential citizen.

PROPOSALS

² <https://github.com/nexusdev/dappsys>

A **proposal** in the governance standard is a suggested action (with suggested arguments) that has not yet been executed. In this sense, the action is being proposed, and the governance standard can define any number of potential paths and states through which it can go through to ultimately be rejected, or accepted and executed.

PROPOSAL STATES

A **proposal state** represents a proposal's status within the governance process. Every action has two states by default: 'rejected' and 'approved'. A rejected proposal is a proposal which is dead, and can no longer change states. An 'approved' proposal is a proposal which will be executed on the blockchain. In Verity's governance standard, any number of arbitrary states may be created representing arbitrarily complex governance rules.

For instance, in the US government, when the action proposed is "create bill" (and the bill itself is the argument), the proposal would go through the "senate vote" state, the "house vote" state, and the "presidential veto period" state, before finally being accepted or rejected.

ROLES

A **role** in the governance standard is a specific function that can be assigned to any agent in the system. For instance, roles in the US government might include Citizen, President, Senator, and Congressman. Other non-obvious roles might include The Constitution, The Senate, and the House; these roles would be filled by smart contracts, which themselves could implement their own governance rules.

Verity's standard also supports hierarchical roles. A child role will inherit all permissions from its parent role unless otherwise specified, as well as any additional permissions assigned to it. For instance a CEO can inherit at the very least all the permissions that a manager has, and a manager can inherit all the permissions of an employee.

PERMISSIONS

Permissions in Verity are given to specific roles and give that role the ability to participate in certain ways within the governance system. There are only two types of permissions within the governance system; roles can be given the permission to execute an action, and they can also be given the permission to change a proposal from one specific state to another specific state.

In the US government, a "senator" role might have permission to execute a "vote in the senate" action, and a "the senate" role might have permission to change a "create bill" proposal from the "senate vote" state to the "house vote state" or the "rejected" state.

GOVERNANCE SYSTEMS

While Verity can represent any governance system (that can be represented in a Turing complete language), it will come with several options out of the box, which make unique use of Verity's own reputation metrics. These

reputation-based governance methods represent new paradigms that can work equally well in traditional settings, as well as decentralized applications.

VALUE-BASED HIERARCHY

In a **value-based hierarchy**, every role in an organization is matched with a value tag, such as “good customer support person,” “good customer support manager” or “good engineer.” The roles exist in a hierarchy, such that executives exist above customer support managers, and customer support managers exist above customer support employees.

The central idea behind value-based hierarchy is that each stage in the hierarchy gets to choose the start-set for the role below it. Thus, executives choose exemplary examples of managers, managers choose exemplary examples of employees, and so on. When someone’s value score rises above zero for a given role, they are automatically given all the permissions and perks associated with that role.

This allows the entire organization to participate in the promotion process and be on the lookout for ideal employees, while still giving higher-ups the ability to choose what type of qualities they’re looking for in each role, and prizing the opinions of agents already in the role.

CONTRIBUTION-BASED VOTING

In **contribution-based voting**, agents vote on actions, and their votes are weighted by the amount of contribution tokens that agent holds within that community. This can be seen as a meritocratic system in which those whom offer more value into the community get more say within it. This voting method can optionally have contribution tokens’ weight decay logarithmically over time, such that people whom have given more recent contributions are given more weight than those whom gave contributions long ago in the organizations history.

SKILL-BASED REPUTARCHY

Reputarchy uses Verity’s reputation and community mechanisms to give decisions to those whom have proven they’re best at making them. If futarchy is described “vote on values, bet on beliefs,” (Hanson, Shall We Vote on Values, But Bet on Beliefs?, 2003) then reputarchy could be described as “vote on values, crowdsource beliefs.” The community decides on a global utility function, and then decisions are made by creating Verity contests that maximize that utility function. Optionally, contractors can apply to enact those decisions, and their proposals will be ranked as well using Verity contests.

In reputarchy, every community chooses a utility function. Then contests are created which measure various decisions related to that

REPUTARCHY IN ACTION

Imagine an “end global warming” community that made a utility function with two variables: how much carbon emissions are lowered within five years, and how ethical each action is considered by ethical members of the community. It would then choose guilds and shared skills which it thought it might need, such as choosing the

“Washington lobbyists” guild skill for the “lobbying” skill tag and the “high-tech inventors” shared skill for the “technology development” skill tag. The community could then create a contest for the best lobbying strategy that met its utility function of lowering carbon emissions, and have the inventors work together to come up with potential promising approaches to reducing carbon emissions. It would choose those actions which had the highest overall reduction in carbon emissions, and hire contractors to execute on those plans.

VERITYDAO

The **VerityDAO** is the organization which oversees running the central protocol and the growth and maintenance of the entire Verity ecosystem. VerityDAO plans to start with a simple multisig governance mechanism, informed by Reputarchy style suggestions, then later switch to fully Reputarchy-controlled governance mechanism with contribution-based rewards as the ecosystem of agent communities matures.

MAKING A PROFIT

VerityDAO makes money by taking a small cut of the reward from every paid contest in every community. This reward then goes into the Verity smart contract, where it can be used to help fund new communities, bolster existing communities, create better infrastructure, or be passed on to agents in the form of weekly payments.

VERITYDAO GOVERNANCE ACTIONS

As the parent organization, VerityDAO is in charge of taking actions that cause the ecosystem to flourish. What follows is a list of some of the most important actions that VerityDAO can take.

SETTING VERITY TOKEN ISSUANCE RATE

Issuance of new tokens in Verity is set at a steady monthly limit, which can be tweaked by the stakeholders. This is analogous to a central bank which is incentivized to maximize the value of the tokens. Newly issued tokens are split percentage wise among all the communities, and are created to fuel Novice Matches and allow the onboarding of new agents into Verity communities.

SETTING VERITY TOKEN BUY AND BURN RATE

Just as it may be prudent to issue more tokens in order to onboard more agents onto the platform, it may also make sense to shrink the supply in order to raise the value of Verity tokens. For this purpose, the VerityDAO has an ongoing offer to buy and burn tokens at a price set by the DAO's governance mechanism. While this price will typically be set to 0, it may be raised above market value if the supply is growing too quickly, in order to incentivize burning of tokens and a reduction in the supply.

CHOOSING CORE VALUES

Like any community that holds values-based reputation, the VerityDAO is in charge of choosing its own core values. These core values will in turn determine the value-weight of all votes in the organization, based on voters' affinity scores with VerityDAO's core values.

ADDING AND REMOVING MEMBERS FROM THE START SET

For each of the core values, the VerityDAO can choose members whom they view as paragons of those values, and add them to the start set. If it comes to light that a person has a different character than previously assumed, they can be removed from the start set, also according to the governance protocol.

SETTING CONTEST FEES

Every contest on the Verity platform has a small percentage fee taken from it, which goes to the VerityDAO. The VerityDAO itself sets this fee at such a level to be competitive, while still allowing maintenance and growth of the platform.

SETTING CONTRIBUTION-WEIGHTED PAYMENT

Every week, a small amount of earnings are taken from the VerityDAO and issued to stakeholders, weighted by their contribution tokens. The VerityDAO votes on this amount.

UPDATING THE PROTOCOL AND GOVERNANCE RULES

The VerityDAO is in charge of creating updates to the protocol, as well as to its own governance rules.

SENDING MONEY

The VerityDAO can send money to any account on the Ethereum platform. This may be in payment to a contractor, as a stimulus to an agent community, or for any other reason.

COLLABORATION LAYER - OBJECTS IN THE VERITY ECOSYSTEM



Figure 12 - Collaboration Layer

Taking a broad overview, the Verity Platform provides the underlying Governance and Reputation infrastructure to build, operate and scale advanced highly collaborative communities, made up of diverse agents. The **collaboration layer** of Verity includes systems and standards to create, read, update and delete agents, content and communities within the ecosystem.

AGENTS

An **agent** in Verity is any entity that can perform actions in the system. Agents can be assigned roles and take actions within a given governance system, and receive scores and rate other agents within a given reputation system; both the reputation and roles of an agent are specific to the communities which they are a part of.

Agents in Verity consist of a public key tracked entity in a single smart contract. The Verity protocol makes no assumptions about who controls the private key, nor any assumptions that tie the public key to real world identities. Instead, Verity allows any identity and key management system to layer on top of its governance and reputation protocols, allowing for a variety of identity solutions depending on the use case of a particular community, governance system, or reputation protocol.

The Verity protocol is specifically engineered such that any part of the system that can take actions (including communities and VerityDAO) are automatically registered in the agent registry. This ensures interoperability between communities, and allows for future integrations where for instance communities themselves can create meta-governing bodies and receive reputation.

CONTENT

Content in Verity refers to any entity in the Verity ecosystem which can be rated or acted upon, but which itself can't take actions. By creating a central repository for content, one can link an individual's view on a piece of

content across communities, and aggregate the opinions of various communities of experts on a single idea or action. Semantic information about the content can be created using the attestation or reputation standards, or can be layered on top using other systems like IPFS's IPLD³. For instance, a single legal policy can be added to the content database as a piece of content, and using attestations, a tag cloud can be created that shows that it is a piece of legal content. Then using attestations and reputation, experts on climate change, foreign policy, and social justice can all separately weigh in from their respective communities on the impacts to their area of interest. Because of the central content registry, these opinions and forecasts can be aggregated into a single user interface. The same can be done with reviews and ratings of things like restaurants.

Content tracking in Verity is expressed as a single searchable smart contract that allows the registry of content. The content can be of four types:

1. An immutable piece of data stored directly in the smart contract.
2. A mutable piece of data editable by an agent and stored directly in the smart contract.
3. An immutable hash/link pair that links out to an off-chain piece of content and can be checked against the hash to make sure it has not changed.
4. A mutable agent/link pair that links out to a mutable piece of content signed by the owner agent to ensure its legitimacy.

The latter two types of content can link out to private sources, and the former two types of content can be encrypted, allowing for various levels of private content. Together, these four types of content can cover nearly all use cases. The Verity protocol is specifically engineered such that all non-active entities within the system (such as attestations and value tags) are automatically registered in the content database, allowing these items to also be rated and given reputation throughout the system.

COMMUNITIES

Communities are places where agents can gather, converse, and work together to provide value to each other or to clients. Every community has different goals and purposes, defined by the agents that are part of that community, and the core values of the community.

CREATING A COMMUNITY

Communities are created in stages. First, the idea for a community is proposed. Potential participants in the community will begin to interact with each other, tagging each other with value tags as they discuss the community.

As the set of ideas formulate into actionable purpose and mission, a group of founding members (or a single

³ <https://github.com/ipld/specs/tree/master/ipld>

founder) will initialize the community by choosing a governance mechanism, initial reputation mechanisms and applicable core values. Core values can be chosen by looking at commonly tagged values between founding members, or can be ignored and uniformly chosen by the founder(s).

After the initialization period, the community actively begins performing work toward their mission. To accomplish initial startup activity, community members begin creating and participating in contests. At this point, self-governance community rules, values and behavior reinforce and amplify the desired community practice toward their mission. As needs arise, community practice can evolve or be tailored according to chosen or amended community governance protocols.

As ongoing practice progresses, community members further interact in contests and campaigns performing work toward their mission which reinforce and amplify metrics in **values-based reputation**, **skills-based reputation** and **contribution-based reputation**. New Verity tokens can enter the community by transforming according to the normal token transformation rules explained above.

COMMUNITY GOVERNANCE ACTIONS

CHOOSING A UTILITY FUNCTION

The community is in charge of choosing its own utility function. This will be used when creating contests, and also for tracking contributions of agents to the community.

CHOOSING CORE VALUES

The community is in charge of choosing its own core values. These core values will in turn determine the value-weight of all votes in the parent organization, based on those voters' affinity scores with the organization's core values.

ADDING AND REMOVING MEMBERS FROM THE START SET

For each of the core values, the community can choose members whom they view as paragons of those values, and add them to the start set. If it comes to light that a person has a different character than previously assumed, they can be removed from the start set.

SETTING CONTEST FEE

Every contest in the community has a small percentage fee taken from it, which goes to the community. The community sets this fee at such a level to be competitive, while still allowing maintenance and growth of the community.

CHOOSING CONTEST TYPES AND SKILLS

Every community can choose skill tags it would like to track, and choose whether they would like to use existing measures or create their own. If they choose to measure the skills themselves, the community must also choose the contest types it will allow to use those skills.

UPDATING CONTEST AND GOVERNANCE RULES

The community can choose to update its privileged contests to new versions, as well as changing its governance contract.

SENDING MONEY

The community can send money it owns to any account on the Ethereum platform. This may be in payment to a contractor, or for any other reason.

USERS AND APPLICATIONS LAYER

The Verity **users and applications layer** refers to the ecosystem of mobile apps, decentralized apps, and web apps that implement the Verity protocol. Together, these applications form a web of communities, focused around a variety of aims, that can interact with each other through common governance protocols and create shared definitions of reputation.

The Verity protocol aims to be application and business model agnostic, working equally well with mobile, desktop, and web protocols (as well as future platforms like VR and AR). Verity communities can be global or localized, public or private, centralized or decentralized. By building on the Verity protocol, these diverse applications can still interact with each-other and benefit from ecosystem innovations. Developers may onboard subsets or whole layers of the Verity protocol within their existing projects or A/B market testing activities.

Fundamental properties built-into the Verity ecosystem are a) credible reputation metrics, b) flexible governance settings and c) all agent, content and community specific central repository data is internally self-generated and maintained within the Verity ecosystem. Within the Verity protocol, the responsibility of maintaining proper incentives and managing “sensitive” user data is offloaded to the protocol itself, turning everyday behavior into verifiable metrics, while allowing applications to focus on core functions related to their purpose. The benefit is happy users aligned with clearly defined and accepted governance, faster time to market (for communities and applications), reduced burden of community management (applications no longer need to build and maintain this on their own), while expanding access to interconnected tools, processes and networks.

EXAMPLE APPLICATIONS

DECENTRALIZED APPLICATIONS

Decentralized applications (Dapps) have a particular problem that other applications don't have. They often need to make decisions that require human judgement, but giving the decision to any one person or small group of people can end up re-centralizing that app by concentrating power. With Verity, all of these human judgements

can be made in decentralized way. For instance, one can **moderate content** without a centralized moderator, **settle agent disputes** without a single company to handle support, **identify trusted agents** for distributed work, and even create **truth oracles** that provide trusted data without relying on a single source.

RATING AND GRADING

Ratings and grades are notorious for creating adverse incentives that cause bad behavior. The centralized review site Yelp has been accused of extorting business to bury negative reviews⁴, and teachers have been known to give their students the correct answers on standardized tests⁵. With Verity, you can **democratize education** through decentralized assessments, create **cheaper insurance** through crowdsourced risk assessments, and **disintermediate review websites** with reviews and ratings that can't be gamed.

COLLABORATIVE CROWDSOURCING

Anything that can be created by an agent on a computer, can be created by Verity crowdsourcing. With Verity, the things built are optimized to meet your exact goals or preferences, and are typically better results than you would receive by contracting a single agent. You can **have software reviewed by thousands of reviewers**. You can create a **compelling creative book**, and give royalties based on how much each person contributes. You can even **design cars, buildings or organizations**, all at a cheaper price point and likely greater quality than an individual agent.

PORTABLE REPUTATION

With Verity, you can earn reputation once, and take it everywhere. This means that **low quality internet comments will be buried**, as the best commenters earn their reputation and have the means to consistently present their reputation from site to site. Instead of needing to prove yourself on each new **marketplace**, you can take it with you (for example transfer reputation from Amazon to eBay). Finally, the **sharing economy** will benefit from higher credential confidence regarding safety and quality of work protections. Broader visibility into Uber, Airbnb, and Snaggoods reputation will all be linked, allowing every agent to get consistent, reliable and high quality service they expect.

DECISION MAKING

With Verity, making decisions will be decision-theoretically optimized according to your goals. Want to get **agent advice** on buying shoes that look great and last forever? With Verity, the crowd will find the perfect pair for you that balances those two goals. **Business strategy** and market needs analysis will forever be changed as companies find out that their CEOs' decisions can be matched or beaten by communities, with lower price points. Eventually,

⁴ <http://www.huffingtonpost.com/news/yelp-extortion/>

⁵ <http://abcnews.go.com/US/atlanta-cheating-178-teachers-administrators-changed-answers-increase/story?id=14013113>

countries may even jump on the reputarchy bandwagon (some noteworthy parties and elections may need improvement to gain alignment with their respective constituents).

FORECASTING

Knowing future possibilities is crucial to wellbeing of millions of people. Wouldn't it be great to know the likelihood of a major war springing up in the next year or the chance of an earthquake in California in the next six months? With Verity, **large events**, **natural disasters**, and **existential risks**, can all be given precise probabilities. These probabilities have been shown to outcompete even the top forecasting agents, and are of similar quality to prediction markets.

MACHINE LEARNING AUGMENTATION

Training machine learning systems to interact with human systems is a common goal, but the existing approaches suffer from limited training data and poor specification of human values. Verity can solve these problems and create a synergistic effect between machines and humans. Using the Verity protocol, for the first time there's an **API for human ingenuity**, which communities can be plugged into in order to solve problems machine learning algorithms can't. The machine learning algorithms can then use these **contests as training data**, significantly reducing the effort needed to compile such a data set. Finally, these machine learning algorithms can be used to further assist humans, completing the cycle. In general, it is assumed that participants in Verity communities can be human, AI or humans augmented by AI.

CONCLUSION

For the first time, Verity makes possible the self-governed generation of credible and immutable reputation metrics that represent attributes and work people care about in the real world. Our Value Rank algorithm allows any humans' values, skills and contribution to be quantified using the notion of transitive trust. Our Verity tokens allow any humans' skills and contributions to a community to be quantified using basic decision-theoretic criteria. By combining this reputation with a novel governance method for flexible crowdsourced contests, communities can harness their own wisdom of the crowd to make highly-informed group decisions, outsmarting expert agents while operating according to an egalitarian reputarchy. This has far reaching implications for governance, decision making, forecasting, crowdsourcing and society at large.

The Verity platform aims to fundamentally redefine trust and collaboration on the internet, allowing users and applications to work together towards shared values and goals, govern themselves, and share in the profit their platforms earn. Verity enables social applications which depend on verifiable, trustworthy and collaborative communities. Applications benefit from happy users aligned with clearly defined and accepted governance, faster time to market, reduced burden of community management and expanded access to interconnected tools, processes and networks. Users and citizens of online communities now have a method to measure and award peer-to-peer, decentralized trust.

BIBLIOGRAPHY

- 99designs. (2016, October 28). *How to become a Platinum designer on 99designs*. Retrieved from 99designs Blog: <https://99designs.com/blog/inside-99designs/platinum-designer-99designs/>
- Aggregative Contingent Estimation*. (2016, September 30). Retrieved from IARPA: <https://www.iarpa.gov/index.php/research-programs/ace>
- Atanasov, P. D., Rescober, P., Stone, E., Swift, S. A., Servan-Schreiber, E., Tetlock, P. E., . . . Mellers, B. (2015, October 8). Distilling the Wisdom of Crowds: Prediction Markets versus Prediction Polls. *Management Science*.
- Baron, J., Mellers, B. A., Tetlock, P. E., Stone, E., & Ungar, L. H. (2014). Two reasons to make aggregated probability forecasts more extreme. *Decision Analysis*, 11(2), 133-145.
- Buterin, V. (2014, March 28). *SchellingCoin: A Minimal-Trust Universal Data Feed*. Retrieved October 17, 2016, from Ethereum Blog: <https://blog.ethereum.org/2014/03/28/schellingcoin-a-minimal-trust-universal-data-feed/>
- Culbertson, S. S., Henning, J. B., & Payne, S. C. (2013). Performance Appraisal Satisfaction: The Role of Feedback and Goal Orientation. *Journal of Personal Psychology*, 189-195.
- Fan, X., Liu, L., & Li, M. (2012). EigenTrust++: Attack resilient trust management. *2012 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)* (pp. 416-425). Curran Associates.
- Galton, F. (1907). Vox Populi. *Nature*, 450-451.
- Google Scholar Search. (2016, October 30). Retrieved from Google Scholar: https://scholar.google.com/scholar?ion=1&espv=2&bav=on.2,or.r_cp.&biw=1366&bih=667&dpr=1&um=1&ie=UTF-8&lr&cites=4085997296011241419
- Gun Sirer, E. (2016, October 17). *Micropayments an Cognitive Costs*. Retrieved from Hacking, Distributed: <http://hackingdistributed.com/2014/12/31/costs-of-micropayments/>
- Hanson, R. (2003). Shall We Vote on Values, But Bet on Beliefs? *Journal of Political Philosophy*.
- Hanson, R. (2012). Logarithmic markets coring rules for modular combinatorial information aggregation. *The Journal of Prediction Markets*, 1(1), 3-15.
- Jose, V. R., Nau, R. F., & Winkler, R. L. (2008). Scoring rules, generalized entropy, and utility maximization. *Operations Research*, 56(5), 1146-1157.
- Kamvar, S. D., Schlosser, M. T., & Garcia-Molin, H. (2003). The eigentrust algorithm for reputation management in p2p networks. *Proceedings of the 12th international conference on World Wide Web*. ACM.

- McGregor, J. (2016, 12 14). *Reddit For Sale: How We Made Viral Fake News For \$200*. Retrieved from Forbes:
<https://www.forbes.com/sites/jaymcgregor/2016/12/14/how-we-bought-reddit-for-200/>
- Nesta. (2016, November 11). *The History | Longitude Prize*. Retrieved November 11, 2016, from Longitude Prize:
<https://longitudeprize.org/history>
- Resnick, P., Zeckhauser, R. J., Swanson, J., & Lockwood, K. (2006). The Value Of Reputation On Ebay: A Controlled Experiment. *Experimental Economics*, 79.
- Spiegel, A. (2016, September 30). *So You Think You're Smarter Than A CIA Agent*. Retrieved from NPR:
<http://www.npr.org/sections/parallels/2014/04/02/297839429/-so-you-think-youre-smarter-than-a-cia-agent>
- Szabo, N. (1999). Micropayments and Mental Transaction Costs. *2nd Berlin Internet Economics Workshop*.
- Sztorc, P. (2016, September 12). *Oracles are the Real Smart Contracts*. Retrieved from Truthcoin Bog:
<http://www.truthcoin.info/blog/contracts-oracles-sidechains/>
- Traupman, J. (2007). *Robust Reputations in Peer-To-Peer Markets*. University of California, Berkeley, Computer Science Division.
- Vogelsteller, F. (2016, September 30). *ERC: Token standard*. Retrieved from Github:
<https://github.com/ethereum/EIPs/issues/20>

APPENDIX A – MATH FORMULAS AND ALGORITHMS

The following sections denote some of the main algorithms needed to implement the Verity protocol. To facilitate ease of reading, the following symbols will be used consistently represent different concepts in Verity.

Definitions	
\mathbb{A} : The set of all agents	α : A specific agent
\mathbb{C} : The set of all communities	c : A specific community
\mathbb{V} : The set of all values	v_c : A specific value within a community
\mathbb{Z} : The set of all skills	ζ_c : A specific skill within a community
τ : Some amount of skill tokens	

A.1 – EIGENTRUST ALGORITHM

A.1 – Definitions	
$loctrust(v_c, \alpha_i, \alpha_j)$: local trust measure of transactions agent i had with agent j for value v_c	$starttrust(\alpha_i, v_c)$: the start trust put into an individual agent i for value v_c from the set of trusted peers
$sat(v_c, \alpha_i, \alpha_j)$: number of satisfactory transactions agent i had with agent j for value v_c	$normloc(v_c, \alpha_i, \alpha_j)$: the normalized local trust value that agent i assigns to agent j for value v
$unsat(v_c, \alpha_i, \alpha_j)$: number of unsatisfactory transactions agent i had with agent j for value v_c	$proptrust(v_c, \alpha_i, \alpha_k)$: the propagated trust value that agent i assigns to agent j for value v_c
	\mathbb{A}_{pv_c} : the start-set of pretrusted agents p for value v_c

1. A **local trust value** is computed for every agent the originating agent interacts with. This is done by taking the number positive interactions with that agent minus the number of negative interactions.

$$loctrust(v_c, \alpha_i, \alpha_j) = sat(v_c, \alpha_i, \alpha_j) - unsat(v_c, \alpha_i, \alpha_j)$$

2. A distribution is defined in which each member of the **start-set** is given equal trust.

$$starttrust(v_c, \alpha_i) = \begin{cases} \frac{1}{|\mathbb{A}_{pv_c}|} & \text{if } \alpha_i \in \mathbb{A}_{pv_c} \\ 0 & \text{otherwise} \end{cases}$$

3. The local trust score is normalized over all local trust scores of agents that this agent has rated, such that all the local trust scores sum to 1, with any trust scores below 0 being dropped entirely. If the normalized trust score is 0, the agent will spread its trust among the start set.

$$normloc(v_c, \alpha_i, \alpha_j) = \begin{cases} \frac{\max(loctrust(v_c, \alpha_i, \alpha_j), 0)}{\sum_{\alpha_j} \max(loctrust(v_c, \alpha_i, \alpha_j), 0)} & \text{if } \sum_{\alpha_j} \max(loctrust(v_c, \alpha_i, \alpha_j), 0) \neq 0 \\ starttrust(v_c, \alpha_j) & \text{otherwise} \end{cases}$$

4. These normalized local trust scores are then propagated, by assuming that trust is transitive. For an agent k that agent i has never met, agent i will add the normalized trust scores of the set of all agents j whom have met agent k , and weight those scores by agent i 's own trust in agent j .

$$proptrust(v_c, \alpha_i, \alpha_k) = \sum_{j=1}^{|\hat{A}_{ik} \cap v_c|} normloc(v_c, \alpha_i, \alpha_j) \cdot normloc(v_c, \alpha_i, \alpha_k)$$

5. This process is continued outward, such that agent k becomes the new agent j . Eventually, the agent has a complete view of every agent in the network. This can also be viewed as a probabilistic process, in which following agents with higher t 's will result in landing on a more trustworthy peer. This markov process can be modeled using linear algebra.

A.2 – EIGENTRUST++ ALGORITHM

A.2 – Definitions

\hat{A}_{ijkv_c} : The set of agents k that the two agents i and j have both independently rated on value v in community c

$avgtrust(v_c, \alpha_i, \hat{A}_{ijkv_c})$: The average amount of trust that agent i places on agents in the set \hat{A}_{ijkv_c}

$|\hat{A}_{ijkv_c}|$: The number of agents in set \hat{A}_{ijkv_c}

\hat{A}_{iM} : The set of agents m that have been rated by agent i

$feedsim(v_c, \alpha_i, \alpha_j)$: The feedback similarity that node i assigns to node k

$feedcred(i, j)$: The feedback credibility that peer i assigns to peer j

$loctrust(i, j)$: The normalized feedback credibility that peer i assigns to peer j

$normloc(i, j)$: As defined in [A.1 – EigenTrust Algorithm](#)

$edgeweight(i, j)$: Chance that peer i will propagate its trust to peer j

1. Find the set of agents \hat{A}_{ijkv_c} that agents α_i and α_j have both rated.

$$\hat{A}_{ijkv_c} \subseteq \hat{A}_c$$

$$\alpha_k \in \hat{A}_{ijkv_c} \text{ where } loctrust(v_c, \alpha_i, \alpha_k) \text{ and } loctrust(v_c, \alpha_j, \alpha_k) \text{ exist}$$

2. Average all local trust ratings together, between all agents k that agents i and j have both rated.

$$avgtrust(v_c, \alpha_i, \hat{A}_{ijkv_c}) = \frac{\sum_{k=1}^n proptrust(v_c, \alpha_i, \alpha_{kn})}{n}$$

$$avgtrust(v_c, \alpha_j, \hat{A}_{ijkv_c}) = \frac{\sum_{k=1}^n proptrust(v_c, \alpha_j, \alpha_{kn})}{n}$$

3. Compute the similarity between two agents i and j by computing the sample standard deviations of all agent ratings that they have in common.

$$feedsim(v_c, \alpha_i, \alpha_j) = 1 - \sqrt{\frac{\sum_{k=1}^n (avgtrust(v_c, \alpha_i, \hat{A}_{ijkv_c}) - avgtrust(v_c, \alpha_j, \hat{A}_{ijkv_c}))^2}{|\hat{A}_{ijkv_c}|}}$$

4. Divide similarity among all agents m that have a similarity score from agent i to create a normalized feedback score.

$$normfeed(v_c, \alpha_i, \alpha_j) = \frac{feedsim(v_c, \alpha_i, \alpha_j)}{\sum_{m=1}^{|\hat{A}_{iM}|} sim(\alpha_i, \alpha_m)}$$

- Define feedback credibility as a metric that weights each normalized trust score $normfeed()$ by each feedback similarity score $feedsim()$.

$$feedcred(v_c, \alpha_i, \alpha_j) = normfeed(v_c, \alpha_i, \alpha_j) \cdot feedsim(v_c, \alpha_i, \alpha_j)$$

- Redefine normalized trust value $normloc()$ to include feedback credibility.

$$normloc ++ (v_c, \alpha_i, \alpha_j) = \begin{cases} \frac{\max(feedcred(v_c, \alpha_i, \alpha_j), 0)}{\sum_{m=1}^{R(i)} \max(feedcred(v_c, \alpha_i, \alpha_m), 0)} & \text{if } \sum_{m=1}^{R(i)} \max(feedcred(v_c, \alpha_i, \alpha_m), 0) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

- Redefine propagated trust naively using the new definition of normalized local trust given above.

$$naiveproptrust ++ (v_c, \alpha_i, \alpha_k) = \sum_{j=1}^{|\hat{A}_{ikjv_c}|} normloc ++ (v_c, \alpha_i, \alpha_j) \cdot normloc ++ (v_c, \alpha_i, \alpha_k)$$

- Set the edgeweight for propagation such that it includes both similarity and normalized trust, putting 85% of the weight on similarity.

$$edgeweight(v_c, \alpha_i, \alpha_j) = 0.15 \cdot normloc ++ (v_c, \alpha_i, \alpha_j) + 0.85 \cdot feedsim(v_c, \alpha_i, \alpha_j)$$

- Set a random threshold between 0 and 1 for propagation using the edge weight.

$$\begin{aligned} \hat{A}_{ikj_w v_c} &\subseteq \hat{A}_{ikj v_c} \text{ where } edgeweight(v_c, \alpha_i, \alpha_j) \geq w \\ w &= \text{Random}(0,1) \\ proptrust ++ (v_c, w, \alpha_i, \alpha_k) &= \sum_{j=1}^{|\hat{A}_{ikj_w v_c}|} normloc ++ (v_c, \alpha_i, \alpha_j) \cdot normloc ++ (v_c, \alpha_i, \alpha_k) \end{aligned}$$

- As with EigenTrust, this can be viewed as a probabilistic process. Take the left principle Eigenvector as with EigenTrust.

A.3 – RELATIVE RANK ALGORITHM

A.3 – Definitions

\hat{A}_{v_c} : The set of agents in community c with value v

\hat{A}_{Nv_c} : The subset of all agents in \hat{A}_{v_c} that aren't in the start set \hat{A}_{pv_c}

A_{Nfv_c} : The subset of all agents in \hat{A}_{Nv_c} that have f feedbacks.

$toprated(A_{Nfv_c})$: Returns the highest rated agent in A_{Nfv_c}

α_{iNfv_c} : The i th agent in set A_{Nfv_c}

$proptrust(v_c, \alpha_{iNfv_c}, \alpha_k)$: as defined in [A.1 – EigenTrust Algorithm](#)

$rankslope(\hat{A}_{Nv_c})$: The slope of the line for all top-rated pairs plotted against how many feedbacks those agents have

\hat{A}_{Fv_c} : The set of all sets A_{Nfv_c}

$rankintercept(\hat{A}_{Nv_c})$: The intercept of the line for all top-rated pairs plotted against how many feedbacks those agents have

f_{α_i} : The number of feedbacks f that agent i has

$relativerank(\alpha_{iNv_c})$: The score that an agent i earns using the Relative Rank algorithm

1. Separate start-set from non-start set members by taking the complement of the set of all users ranked with value v in community c .

$$\hat{A}_{v_c} - \hat{A}_{pv_c} = \hat{A}_{Nv_c}$$

2. Separate non-start set agents into sets according to how many f feedbacks each agent has received (both positive and negative)

$$\hat{A}_{Nfv_c} \subseteq \hat{A}_{Nv_c} \text{ where agents have } f \text{ feedbacks}$$

3. Find the top rated agent in every group

$$toprated(\hat{A}) = \max(proptrust(v_c, \alpha_{iNfv_c}, \alpha_k)) \in \hat{A}_{Nfv_c}$$

4. Find a line of best fit for all pairs $(f_n, toprated(f_n, \hat{A}_{Nfv_c}))$, and determine the slope and intercept of that line.

$$rankslope(\hat{A}_{Nv_c}) = \frac{|\hat{A}_{Fv_c}| \cdot \sum_{n=1}^{|\hat{A}_{Fv_c}|} (f_n \cdot toprated(f_n, \hat{A}_{Nfv_c})) \cdot \sum f_n \cdot \sum toprated(f_n, \hat{A}_{Nfv_c})}{|\hat{A}_{Fv_c}| \sum (f_n^2) - (\sum f_n)^2}$$

$$rankintercept(\hat{A}_{Nv_c}) = \frac{(\sum toprated(f_n, \hat{A}_{Nfv_c})) (\sum (f_n^2)) - (\sum f_n) (\sum (f_n \cdot toprated(f_n, \hat{A}_{Nfv_c})))}{|\hat{A}_{Fv_c}| \sum (f_n^2) - (\sum f_n)^2}$$

5. Define a non-start set agent i 's **Relative Rank** as an area relative to that line

$$relativerank(\alpha_i, v_c) = \frac{proptrust(v_c, \alpha_i, \alpha_k) - rankintercept(\hat{A}_{Nv_c})}{rankslope(\hat{A}_{Nv_c}) \cdot f_{\alpha_i}}$$

6. Repeat steps 3 - 5, but for set \hat{A}_{pv_c} .

A.4 – AFFINITY SCORE ALGORITHM

A.4 – Definitions

$valuerank(v_c, \alpha_i)$: A Value Rank function that combines aspects of Eigentrust++ with Relative Rank
 c : The community in which the affinity is being calculated

$affinity(c, \alpha_i)$: The function that calculates an agent i's affinity to community c
 n : The total number of core values within the community.

1. Take the agents Value Rank along all core values for that community, and average them together.

$$affinity(c, \alpha_i) = \frac{\sum_{c=1}^n valuerank(v_c, \alpha_i)}{n}$$

A.5 – TOKEN TRANSFER ALGORITHM

A.5 – Definitions

$\hat{A}_{\tau_{jcm}}$: The set of all agents that hold skill tokens for skill j in community m
 $\tau_{\alpha_i \zeta_{jcm}}$: The amount of tokens τ that agent i holds in skill j in community m
 $naivesim(\zeta_{jcm}, \zeta_{kc_n})$: The similarity between token holders who hold tokens in skill j in community n , and the amount of tokens they hold in skill k in community n
 $skillsim(\zeta_{jcm}, \zeta_{kc_n})$: The naivesim but all negative correlations are set to 0

$naiveexchrate(\alpha_q, \zeta_{jcm}, \zeta_{kc_n})$: The amount of tokens we would expect an agent q to have in skill j given their tokens in skill k
 $exchrate(\alpha_q, \zeta_{jcm}, \zeta_{kc_n})$: The $naiveexchrate$ weighted by similarity
 $exptok(\alpha_q, \zeta_{jcm})$: The amount of tokens we would expect an agent q to have in skill j given their tokens in all other skills
 $exchdeac(\tau_{\zeta_{jcm}}, \alpha_q)$: The algoirhtm that deactivates exchanged tokens based on similarity

1. Take the set of all agents whose tokens τ for skill j in community m are greater than 0.

$$\hat{A}_{\tau_{jcm}} \subseteq \hat{A}_{cm} \text{ where } \tau_{jcm} > 0$$

2. For every agent i in this set, create a pairs $(\tau_{\alpha_i \zeta_{jcm}}, \tau_{\alpha_i \zeta_{kc_n}})$ for the tokens they hold for skills j and k in communities m and n .
3. Calculate a naïve skill similarity score by determining the [Pearson Correlation Coefficient](#) for these pairs. This score determines if the points are correlated.

$$naivesim(\zeta_{jcm}, \zeta_{kc_n}) = \frac{|\hat{A}_{\tau_{jcm}}| \cdot \sum_{i=1}^{|\hat{A}_{\tau_{jcm}}|} (\tau_{\alpha_i \zeta_{jcm}} \cdot \tau_{\alpha_i \zeta_{kc_n}}) \cdot \sum \tau_{\alpha_i \zeta_{jcm}} \cdot \sum \tau_{\alpha_i \zeta_{kc_n}}}{\sqrt{(|\hat{A}_{\tau_{jcm}}| \sum (\tau_{\alpha_i \zeta_{jcm}}^2) - (\sum \tau_{\alpha_i \zeta_{jcm}})^2) \cdot (|\hat{A}_{\tau_{jcm}}| \sum (\tau_{\alpha_i \zeta_{kc_n}}^2) - (\sum \tau_{\alpha_i \zeta_{kc_n}})^2)}}$$

4. For all $naivesim()$ below 0, set the skill similarity score to 0.

$$skillsim(\zeta_{jc_m}, \zeta_{kc_n}) = \begin{cases} naivesim(\zeta_{jc_m}, \zeta_{kc_n}) & \text{where } naivesim(\zeta_{jc_m}, \zeta_{kc_n}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

$\hat{A}_{\tau_{\zeta_{jc_m}}} \neq \hat{A}_{\tau_{\zeta_{kc_n}}}$. This has the effect of correcting the imbalanced similarity rate of large communities, which likely will have many high similarity rates simply due to chance.

5. Use linear regression to create the naïve exchange rate for the two skills.

$$\begin{aligned} naiveexchrate(\alpha_q, \tau, \zeta_{jc_m}, \zeta_{kc_n}) \\ = \frac{|\hat{A}_{\tau_{\zeta_{jc_m}}}| \cdot \sum_{i=1}^{|\hat{A}_{\tau_{\zeta_{jc_m}}}|} (\tau_{a_i \zeta_{jc_m}} \cdot \tau_{a_i \zeta_{kc_n}}) \cdot \sum \tau_{a_i \zeta_{jc_m}} \cdot \sum \tau_{a_i \zeta_{kc_n}} \cdot \tau_{a_q \zeta_{jc_m}}}{|\hat{A}_{\tau_{\zeta_{jc_m}}}| \sum (\tau_{a_i \zeta_{jc_m}}^2) - (\sum \tau_{a_i \zeta_{jc_m}})^2} \\ + \frac{(\sum \tau_{a_i \zeta_{kc_n}}) (\sum (\tau_{a_i \zeta_{jc_m}}^2)) - (\sum \tau_{a_i \zeta_{jc_m}}) (\sum (\tau_{a_i \zeta_{jc_m}} \cdot \tau_{a_i \zeta_{kc_n}}))}{|\hat{A}_{\tau_{\zeta_{jc_m}}}| \sum (\tau_{a_i \zeta_{jc_m}}^2) - (\sum \tau_{a_i \zeta_{jc_m}})^2} \end{aligned}$$

6. Multiply the naïve rate by the similarity score, which lowers the output of the linear regression when it applies less.

$$exchrates(\alpha_q, \zeta_{jc_m}, \zeta_{kc_n}) = naiveexchrate(\alpha_q, \zeta_{jc_m}, \zeta_{kc_n}) \cdot skillsim(\zeta_{jc_m}, \zeta_{kc_n})$$

7. Calculate how many tokens in a particular skill we expect an agent to have, by averaging the exchange rate rate of all skills s they have with that skill.

$$exptok(\alpha_q, \zeta_{jc_m}) = \frac{\sum_{s=1}^{|Z_{\alpha_q}|} exchrates(\alpha_q, \zeta_{jc_m}, \zeta_{\alpha_q s})}{|Z_{\alpha_q}|}$$

8. When exchanging tokens, deactivate all tokens that cause a user to go above their expected tokens for that skill.

$$exchdeac(\tau_{\zeta_{jc_m}}, \alpha_q) = \begin{cases} activated & \text{while } \tau_{\alpha_q \zeta_{jc_m}} \leq exptok(\alpha_q, \zeta_{jc_m}) \\ deactivated & \text{otherwise} \end{cases}$$

A.6 - TOKEN TRANSFORMATION ALGORITHM

A.6 – Definitions

$naivetransfact(\alpha_q, \tau_{\zeta_{jcm}}, \zeta_{kc_n})$: The amount of tokens we should expect to be deactive

$skillsim(\zeta_{jcm}, \zeta_{kc_n})$: As defined in [A.5 – Token Transfer Algorithm](#)

$transfact(\alpha_q, \tau_{\zeta_{jcm}}, \zeta_{kc_n})$: same as $naivetransfact()$ but with limits on when tokens can be transformed.

$affinity(c_n, \alpha_q)$: As defined in [A.4 – Affinity Score Algorithm](#)

$exptok(\alpha_q, \zeta_{kc_n})$: As defined in [A.5 – Token Transfer Algorithm](#)

$transfdeact(\alpha_q, \tau_{\zeta_{jcm}}, \zeta_{kc_n})$: The amount of tokens that we should expect to be deactivated when transforming tokens based on similarity.

1. Define the naïve transformation activation amount as weighted by skill similarity.

$$naivetransfact(\alpha_q, \tau_{\zeta_{jcm}}, \zeta_{kc_n}) = \tau_{\zeta_{jcm}} \cdot skillsim(\zeta_{jcm}, \zeta_{kc_n})$$

2. Check if the user is above their expected tokens, lacks affinity with the community, or the two communities have no similarity. If so disallow transformations. Otherwise, follow the naïve transformation procedure.

$$transfact(\alpha_q, \tau_{\zeta_{jcm}}, \zeta_{kc_n}) = \begin{cases} fail & \begin{array}{l} \tau_{\zeta_{jcm}} + \tau_{\alpha_q \zeta_{kc_n}} \geq exptok(\alpha_q, \zeta_{kc_n}) \\ or\ affinity(c_n, \alpha_q) < 0.5 \\ or\ skillsim(\zeta_{jcm}, \zeta_{kc_n}) = 0 \end{array} \\ naivetransfact(\alpha_q, \tau_{\zeta_{jcm}}, \zeta_{kc_n}) & otherwise \end{cases}$$

$$transfdeact(\alpha_q, \tau_{\zeta_{jcm}}, \zeta_{kc_n}) = \tau_{\zeta_{jcm}} - transfact(\alpha_q, \tau_{\zeta_{jcm}}, \zeta_{kc_n})$$

APPENDIX B – MATHEMATICAL BUILDING BLOCKS IN CONTESTS

All default contest types built in to Verity use essentially the same basic mathematical building blocks. The Client provides a **utility function**, which is a mathematical representation of their preferences. Then, agents try to fulfill those preferences. Their submissions are ranked on each preference using a **probability distribution**, which is a mathematical representation of their uncertainty about how the submissions should be ranked according to the client's criteria. These probability distributions might be over a **binary outcome**, spread over several **categorical options**. Or be trying to pinpoint a number in a **scalar outcome**. The probability distributions are combined using **pooling**, which is a mathematical tool to turn multiple probability estimates into one combined estimate, and oftentimes made more accurate with extremizing. A **monte-carlo method** is then used on these final probability distributions as they fit within the utility function. A monte-carlo method can be thought of as a guess-and-check method that a computer uses. Finally, either a **Bayesian scoring function** or a **divergence measure** is used, comparing the final values to the values each agent came up with. A Bayesian scoring function is a way to compare an agent's performance to some idealized notion of performance over time, and a divergence measure does the same thing, but compares against an ideal distribution instead of a single answer. The results of this function are used to redistribute tokens.

UTILITY FUNCTIONS

A utility function is a mathematical representation of preferences. In decision theory, they're central to the concept of making sound decisions, and for that reason they're the basis of Creativity tokens in Verity. One's creativity will be scored based on how well one can generate options that maximize a given utility function. In addition, for each input into a utility function, a contest will be created to earn Accuracy and/or Creativity tokens.

Because there are near limitless representations of utility functions, Verity must allow for the possibility of arbitrary complexity. Given this constraint, each utility function will be its own smart contract on the Ethereum platform. These smart contracts will have standard functions, which input any number of variables through a contest format, and output a number. The higher the number, the more utility received.

Utility functions are hard to grasp for the average agents, and we're actively looking into easy ways to intuitively create utility functions, including LENS modeling, willingness to pay models, visual equation builders, and premade utility function libraries. It's also worth noting that for many applications, utility functions won't have to be created by the agent, as the utility function will be premade by the front-end dapp, as in our examples with smart-contract security, and as shown in the wild with the 99designs style attributes interface, which provides a simple graphical interface for creating a utility function that describes an ideal design.

SCORING RULES AND DIVERGENCE MEASURES

A scoring rule is a rule which incentivizes participants to be accurate to create good forecasts by giving higher score to more accurate forecasts. A strictly proper scoring rule is any scoring rule in which incentivizes participants to be honest about their true probabilities, by giving the highest score to the most accurate predictions. In 2008, two families of strictly proper scoring rules were found to be able to incorporate decision-theoretic notions of risk-

tolerance, information-theoretic notions of information gain, and approximate all of the popular scoring rules (Jose, Nau, & Winkler, 2008). These constructions were extended to incorporate the property known as sensitivity-to-distance which allowed forecasters whom were closer to the correct answer to receive higher scores, as well baseline distributions, which allowed for the notion of prior information.

The incorporation of a baseline distribution also allows the possibility for emulation of market scoring rules, a creation of Robin Hanson (Hanson, Logarithmic markets coring rules for modular combinatorial information aggregation, 2012) which use previous information gained from forecasters as the baseline.

By incorporating both the discrete and continuous cases of these two constructions, one can allow every contest to use the scoring rule that best suits their needs, whether through a website choosing this option for their agents, or by creating a wizard through which a client can choose the scoring rule that best fits their needs. These constructions also map to a family of divergence measures that can be used for consensus contests.

POOLING AND EXTREMIZING ALGORITHM

The pooling algorithm is the algorithm which combines probability distributions into a single distribution. A pooling algorithm can be as simple as a weighted average, known as linear pooling, with Accuracy tokens used for the weight. There are several other algorithms which are shown to be more accurate by engaging in a process known as extremizing, a process of moving aggregated distributions towards the extremes that has sound theoretical backing (Baron, Mellers, Tetlock, Stone, & Ungar, 2014). An active area of our research is around the tradeoffs to be made in terms of simplicity, gas costs, and accuracy in different pooling and extremizing algorithms.

MONTE-CARLO SIMULATION

A Monte-Carlo simulation for our purposes is a way to brute-force input the results from an aggregated probability distribution into a utility function. It can then be used to show how different submissions rank in regards to that utility function. Monte-carlo methods are very computationally expensive, and difficult to run on the blockchain. For more on how we'll deal with this, please see [Appendix C – Technical Details and Challenges](#).

FULL PAGE VERITY ARCHITECTURE FLOWCHART



EXPENSIVE COMPUTATIONS

Verity has a number of computations that would simply be too expensive to run on-chain. Particularly for Monte-carlo simulations, any decrease in cost and speed creates a corresponding increase in accuracy, so any optimizations made directly impact the quality of the platform. Because of this, it's critical that the costs and speeds of computation are lowered from those typically seen on Ethereum.

On the other hand, the Ethereum blockchain provides availability, accuracy, and immutability guarantees that are critical for an application whose purpose is to quantify trust. It's critical that these promises aren't weakened too severely.

The paradigm that Verity plans to use to balance these two issues is to push expensive computations to off-chain, but to allow complete verification on-chain if there's any disagreements.

This paradigm will be achieved through two emerging technologies: state channels and interactive verification.

STATE CHANNELS

The first technology is state channels. The idea behind state channels is that all parties involved in a transaction must agree on the results of a computation. If they all agree, only the results of the computation are posted on the Blockchain. If any party doesn't agree on the results of the computation, it is then run on-chain to determine whom was correct. An early example of generalized state channels is the concept of "smart state channels" in Raiden.

STATE CHANNELS IN ACTION

One example of where state channels might be used is in determining the payouts for contest rewards and Verity tokens at the conclusion of a contest. If everyone agrees on the proper payouts within some specified time frame, payouts can happen cheaply and instantly, with only a small gas cost needed to update the final distribution of rewards on the blockchain.

If parties do not agree on the proper distribution of tokens, then the payout structure degrades to an on-chain method in which each participant is responsible for claiming their own rewards and therefore updating their own agent state within the smart contract. This method is more costly and time consuming, but its existence ensures that the reputation calculations can always be trusted.

The existence of Verity's value-based reputation disincentivizes "griefing attacks" on the original state-channel based method, as agents whom submit wrong answers in order to raise costs for others will lose reputation, and therefore the ability to have financial stake in the community.

INTERACTIVE VERIFICATION

Interactive verification works similarly to state-channels, but with a twist: Firstly, the computations need not be run by people involved in the transactions. Secondly, the full computation need not be done on chain, instead only enough needs to be run to determine which party is correct.

The way this works is that a computation is run completely off-chain by an interested party who posts up a security deposit. The results of this computation can then be challenged by another party, whom also posts up a security deposit. If challenged, the computation is then verified on-chain, and the incorrect party loses their deposit to the correct party. Early inroads to interactive verification were made with Ethereum computation market. More recently, TrueBit presented a method that works even for computations that would be too expensive for Ethereum's gas limit.

INTERACTIVE VERIFICATION IN ACTION

One place in which interactive verification would work for Verity is in its computation of Monte-carlo simulations. Agents (especially mobile agents) likely won't have the computation power to run the simulations locally, and the full computation will most likely be above Ethereum's gas limit. Both issues rule out state-channels as a viable solution, and thus require an interactive verification approach.

Taking TrueBit's approach, this would involve submitting a Merkle tree of how one arrived at the final expected value of each contest submission (using a pseudorandom seed provided by the blockchain), as well as the expected value itself to the blockchain. A challenger whom thought the final computation was wrong would then be able to interactively challenge agent parts of the merkle tree where they thought the computation was wrong, and the blockchain would verify who was correct.

GAS COSTS

Gas costs on Ethereum are the costs associated with running computations and storing data on all the agents in the Ethereum network. While gas costs can be significantly reduced using the methods mentioned in the Expensive Computations section, they can never be completely eliminated.

This presents a challenge for a platform such as Verity that hopes to have broad appeal, as agents aren't accustomed to paying for internet platforms, and even small costs have significant overhead in terms of the cognitive load (Szabo, 1999). Thus it remains important to enumerate whom will pay for gas, and how that payment will appear as choices for the agent. Verity has separate answers for these questions over the lifetime of the product.

SHORT TERM – MARKET SELECTION AND FOUNDATION SUBSIDIES

In the short term Verity will take two approaches to ensure that gas costs don't prevent adoption of the platform. Firstly, it will make a careful selection of its early markets, specifically targeting agents and developers whom understand the technology, and are willing to pay for the benefits of decentralization and transparency. In practice, this means targeting integration with other decentralized applications on Ethereum, whose agents and developers are both in this category.

Secondly, the Verity Foundation will work with a select group of partners to integrate with their existing crowdsourcing and sharing economy websites and apps, and completely subsidize the gas usage of these organizations for a number of years. Having immediate integration with the “legacy internet” and the existing “app ecosystem” is incredibly important for the future growth of Verity, and these integrations and subsidies will allow Verity to rapidly reach product-market fit within these key markets.

LONG-TERM – GAS AUTOMATION AND WEBSITE SUBSIDIES

In the mid-term, as the reputation tokens begin to gain real-world value and contests with high rewards become common, Verity will work to create a system of smart-contracts that allows the agents to pay for their own gas without conscious interference. By automatically selling tiny fractions of the reputation tokens and rewards that an agent earns on a decentralized marketplace, Verity can create a system where agents are paying for gas costs without ever having to make the conscious decision to reach into their wallet. A system like this is necessary for widespread agent adoption in the long term.

In addition to agents paying their own way (which may not work for agents whom are performing poorly in contests), websites and apps can subsidize agents usage of their website by paying the gas costs themselves, in a similar way that they pay for computations now using a service like Amazon Web Services. At this point, Verity will have a proven use and track record, and can be thought of as a competitive advantage or even a cost-of-doing-business.

RANDOMNESS

One issue with using probabilistic methods like eigenvectors and monte-carlo simulations to determine reputation is that a solid source of randomness is needed that can't be manipulated by agents to gain more reputation. Non-manipulable randomness on the blockchain is hard to come by, and this represents a technical challenge that Verity must solve. To start off, Verity will use a simple semi-trusted oracle for offchain randomness such as OraclizeIt⁶. Over the long term, Verity would like to mix several sources of randomness together from many different actors with different incentives, such as OracleizeIt, Blockhashes, and a variety of RandAOs⁷.

⁶ <http://www.oraclize.it/>

⁷ <https://github.com/randao/randao>