Network Whitepaper

Joel "@fractastical" Dietz

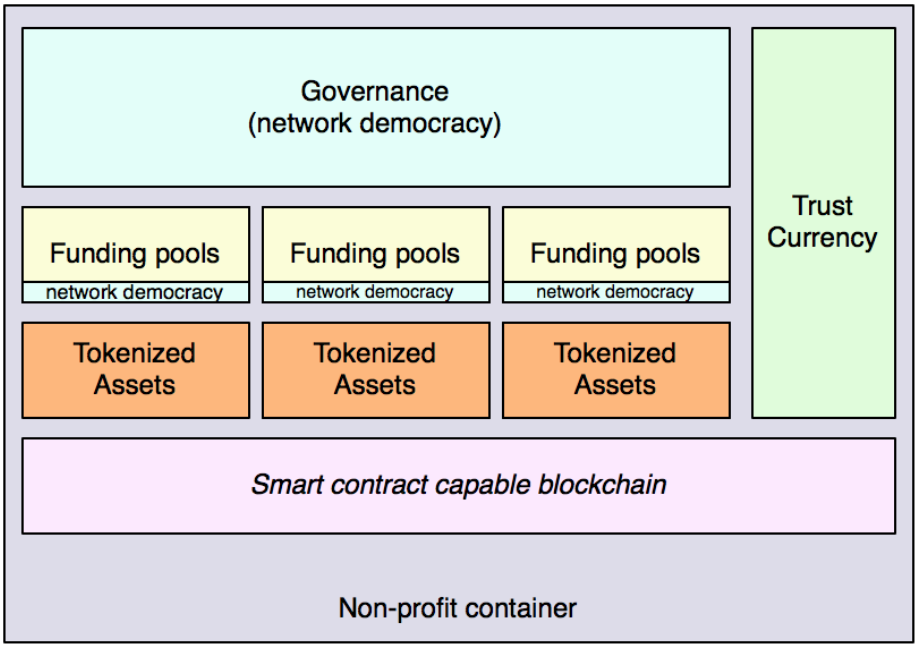
April 19 2017 (anticipated release date)

1 Abstract

The Network Stack proves a properly incentivized basis for maintenance of public goods, especially for new technologies like artificial intelligence that do not fit into any established framework for international governance. This is made possible by a nested design that combines reputational systems and smart-contract capable blockchains. We use this launch our own network implementation called the "Network State."

2 The Network Stack

2.1 Components of the Network Stack



The network stack is a comprehensive way of using the benefits of network organization and deploying them in such a way that they can be used in a variety of industry sectors. In particular, it integrates real world assets with accountability mechanisms in trust currency and in the existing rule of law.

Key components of the Network Stack are network democracy, trust currency, nested holonic organization, and integration into existing legal systems.

2.2 Network Democracy

2.2.1 Why networks?

> The only organization capable of unprejudiced growth, or unguided learning, is a network. All other topologies limit what can happen.
>
> *Kevin Kelly*[1]

An open network has the advantage of not biases based on precognition and evolving along with the constituents of that network. This, in turn, gives people a sense of emotional connection and ownership of the result, and allows them to put a greater than normal amount of effort. Open networks also, rather than presenting a single dominant structural pattern (such as the joint-stock corporation) allow numerous different types of organizations to exist alongside and augment it. Additionally, unlike traditional organizations, the network stack is deliberately designed for automation and participation of AI and other novel optimized decision making mechanisms.

2.2.2 What is network democracy?

Network democracy is a stake-weighted delegated voting system (liquid democracy) which also provides options for enhanced stake and programmable liquidity.

2.2.3 What is stake weighted?

Stake weighted means that voting power is directly proportional to the amount of stake you have in a project. In the blockchain world this is typically called your "tokens" or "coins." In in the world of stock, this would be called your "stock" or "shares."

2.2.4 What is liquid democracy?

Liquid democracy is the same as "delegated voting" and implies that someone can at any point delegate their vote to another party for as long as a period as they desire. This allows them to maintain a degree of control without having to participate in minute decision making. This is in some ways similar to the

decision an investor makes when they give someone capital to work on a project with the key innovation that the investor retains control of the capital.

2.2.5 Why network democracy?

Stake-weighted liquid democracy is a highly flexible governance model which accommodates anything from one-member-one vote to traditional corporate arrangements to novel forms of collectives. It also appropriately incentivizes the many different types of possible contributions to a network (code, community contributions, funding) and allows both decision making capability and rewards. It also integrates with other automated decision making methods such as futarchy.

2.2.6 What is a variable sale price?

Rather than a single round in which all tokens or shares are sold at the same price, the evolving ICO or crowdsale model has been allowing a variety of sale parameters.

This system at least includes the following.

Time-based linear sale model:

$$p = p(t * n)$$

Token price adjusts based on the time of sale.

Token-based linear sale model:

$$p = p(t_s * n)$$

Token price adjusts based on amount of previous tokens sold with a start and end price.

Incentivized discovery model:

$$p = p(t * n)$$

Our ongoing sale makes use of the incentivized discovery and token-based linear sale models.

2.2.7 What is enhanced voting stake ?

Enhanced voting stake makes use of a locking period is a proposed addition to the stake-weighted liquid democracy that does not allow tokens to be traded for a certain period. This can also allow a person to have a proportionally higher voting weight who has a provable long-term stake in the network.

Traditional model:

$$v_w = s$$

Voting weight and network stake are equivalent.

Enhanced model:

$$v_w = s + s(tl * n)$$

In this case, voting weight is no longer simply network stake s but stake plus an additional amount of weight depending on both the length of the time lock and the additional parameter n. For example, with tl denominated in months and an n of .2, a 5 month time lock would effectively double the voting weight.

2.2.8 What is programmable liquidity?

Programmable liquidity indicates that the stake can be gradually introduced to the network for tradability. It is complementary to a locking period, in the sense that you can accommodate both long-term stake and liquidity with incentives on both ends.

$$s_l = s((t - tl)/tl)$$

In this case liquid stake s_l is the total stake owned by the participant multiplied by the remaining time of the time lock as a percentage. This drip liquidity allows people to gradually realize their returns without any major disruption to the network state.

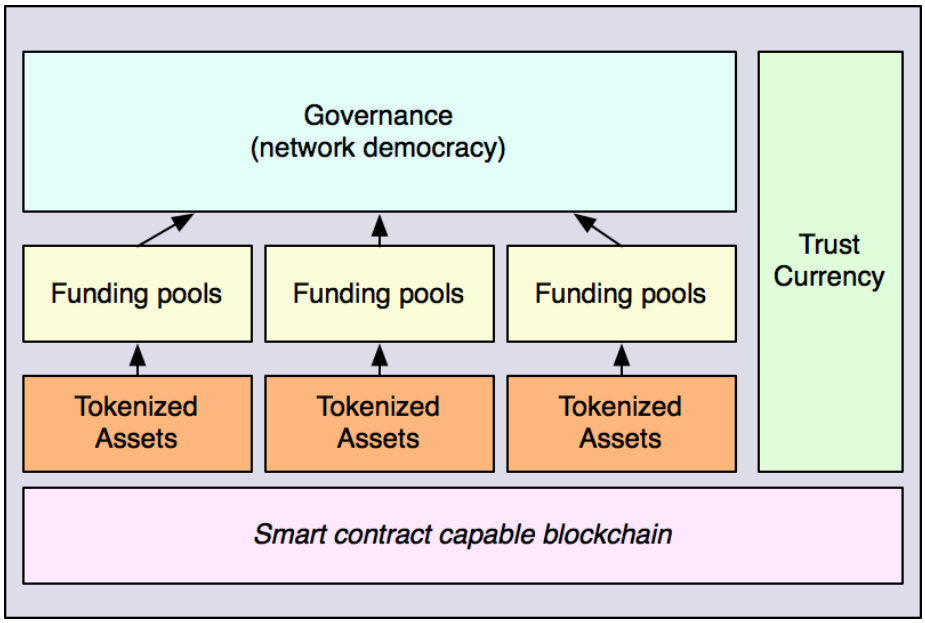
2.2.9 Futarchy for automation

Futarchy that can allow automation of decision making via prediction markets. Network democracy accommodates vote delegation to an automated prediction market.

2.2.10 Enhancements on traditional models

The network stack provides several clear enhancements of traditional models including incentives for early adoption, compatible short and long-term incentives, flexible liquidity models, accommodating multiple organizational types, and accountability and automation via reputational systems.

2.3 Funding in network stack



The network stack, by virtue of nested smart contracts which pass proceeds up to the maintainer of the network stack is able to fully self-fund. This is an evolution in the funding of public goods as neither donations nor taxes are needed in this model.

2.3.1 Funding pools

A funding pool is a organization governed by network democracy that then distributes funds. It can be structured as similar to an investment fund, in which case it seeks profit yielding opportunities or for some other purpose (e.g. advocacy, education). Funds that go into any funding pool can be allocated by a higher level network democracy or received independently.

2.3.2 Tokenized assets

A key element of the network stack is that it maybe be used to govern other real word assets, such as real estate, enterprise SaaS deals, and other tokenizable elements derived from classic investment vehicles. Specific details on these deal opportunities are made available to members of the Network.

2.3.3 Real estate as the ultimate real asset

Real estate is the ultimate real asset, insofar as it has stable and often the appreciating value, lends itself extremely well to subdivision and tokenization, and serves as a backing for the value of the network as a whole.

2.4 How the Network Stack funds itself

Realized returns, which can be from any allocation of funds into areas which return profit or the appreciation of owned assets are subject to certain parameters as established in the governing contracts of the organization. More specifically, a minimum of 1 percent of realized returns are passed up the chain to the next highest organisation. This allows the maintainers of the network stack to achieve an automated income that facilitates the growth of entire network.

2.5 Trust Network

The trust network is made up of many independent trust endoresments (i.e. vectors) which create accountability and allow automation.

2.5.1 Trust equation

$$\sum^{\infty} \vec{i}^{t} = T$$

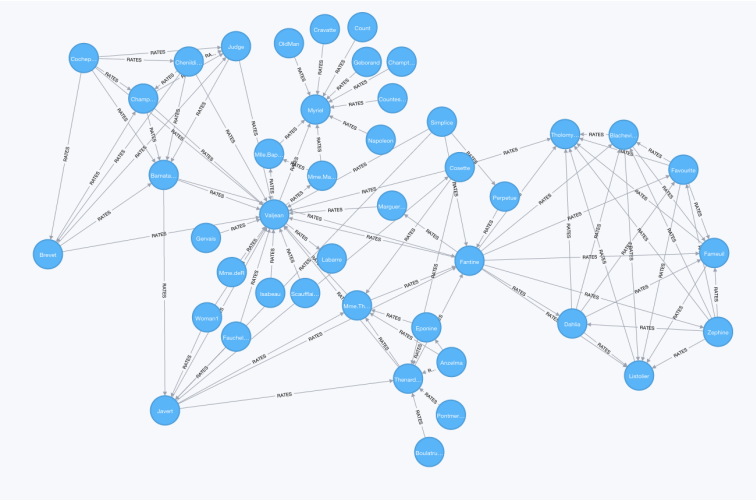
The trust is the sum of all trusted parties and their recommendations from the standpoint of an individual (here represented as vectors which can be positive or negative). In this case, we record all of these endorsement allow multiple levels of ascription and derivation.

2.5.2 Trust vectors

Trust currency is made up by individual immutable trust vectors. Each trust vector $\vec{i}^{t}$ is an endorsement. A endorses B for X. This has a timestamp and an optional weight represented as a number from [-1 .. 0 .. 1]. The ability to provide a negative filter as well as a positive one is important for filtering out signal and noise, including instances like Facebook fake news). As with other open networks, it has an open access policy and the nature of the endorsements is pseudonymous.

2.5.3 The trust graph

These vectors are optimally represented as a graph.



[2]

This graph illustrates that trust (including what we believe to true) can be derived from the various trust vectors. This also allows easy use of data science to create.

2.5.4 Implications of a trust model

A open reputation ledger built of individual trust vectors is a critical element in the Network Stack. It helps incentivize positive overall social contributions and augment the network more generally. It also allows full automation . For example, investment and portfolio allocation decisions can be made by artificial intelligence. It also makes certain other accountability mechanisms less necessary due to the strong social component.

2.6 Holonic nature of the network stack

> No man is an island- he is a holon
>
> *Arthur Koestler*[3]

> The Kosmos is a series of nests within nests within nests indefinitely... holarchies of holons everywhere!
>
> *Ken Wilber*[4]

2.6.1 Key holonic principles

Nested Holonic are looking to realize syncronicity with both larger systems (seasonal cycles, harmonics) and smaller internal systems (human cog-

nition, sleep cycles), of which all organizational types are subject. The optimal design takes into account as many of these as possible and allows them to seemlessly network with each other.

```
(network
    (members
        (human 'alice')
        (human 'bob')))
```

Multiple levels:

```
(network
    (members
        (network 'quantum consortium'
            (members 'Q-HOSS')
            (members 'FutureCorp'))
        (network
            (members 'alice')
            (members 'bob')
        )))
```

Eventually these systems look increasingly like fractals.

Evolving Holonic design insists that the system design does not lock it into a static state but allows for constant vetting and evolution. This is why there cannot be a single 'network state.'

```
(network
    (members
        (human 'alice')
        (human 'bob')
        (other '2501')
      ))
```

Extensible Holonic systems are always looking for opportunities for growth or 'networking,' and any holonic system needs to be designed for this possibility.

```
(network
    (members
        (human 'alice')
        (human 'bob'))
     (reputation trust-ledger))
```

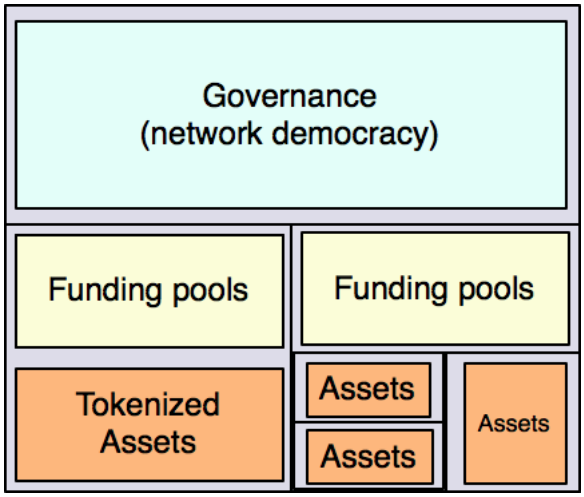
```
(network
    (members
        (human 'alice')
        (human 'bob'))
```

```
(gov
   (tier-1 merkle-dao)
   (tier-2 network-democracy)))
```

Ego-extension What we mean here by ego-extension in this sense is not a chemical or emotive state, it is a name for the principle of networking which allows seamless transition to various states of abstraction within nested systems. In particular, art and music allow people to cultivate states of ego-extension. These can be cultivated in the context of a "metta-org" that serves as an organization specifically dedicated to this purpose. This may be a logical evolution of the ostensible purpose of religion without dogmatic content.

```
(org
    (participants
       (human 'alice')
       (human 'bob'))
    (metta-org (div 1\%))
```

2.6.2 Network stack as a representation of holonic principles



This represents that, both respect to legal integration and governance principles, the stack here may have a near-infinite number of nested sub-entities, each with its own governance principle. With respect to legal systems, this is made possible by the increasing the ease in creating legal entities created by some innovative nation states via APIs and other technologically innovative means. Blockchain and smart contracts also complement this structure by being extensible by design.

2.7 Living Holons

The idea of a "holon" was circulated through the Ethereum community during the lead0-up to the crowdsale and was used to refer to an open living space where ideas could be circulated. Notably, the Palo Alto version of the same hosted Vitalik Buterin, and Anthony Dinofrio of Ethereum present at the first Decentralized Autonomous Society meetup on Jan 10, 2014. There have been 23 meetups on various aspects of decentralized governance since, including notable presentations by Ralph Merkle, DASH team, and associates of Doug Engelbart, as well as numerous other events.[5]

3 Ethereum as a Model

> It would be very nice if there were a protocol whereby unforgeably costly bits could be created online with minimal dependence on trusted third parties.[6]
>
> *Nick Szabo*

3.1 Bitcoin as the Forerunner

Bitcoin proved several interesting points about the viability of a network. First, it proved that a novel incentive mechanism could be use to bootstrap a network with little intrinsic utility outside of its network effect. Second, that a community could grow up around it with a ideological and economic interest in preserving it.

3.2 Ethereum's Incentive model

One major reason for Ethereum's success is that it continued the incentivized open network model of Bitcoin. The code was open source and anyone could participate both in open source code, organization, education, and funding. Contributions also all amounted to substantial returns when the network was released and scaled. It also accommodated a variety of interests, including socially positive, technologically innovative, and economically rewarding.

3.3 Ethereum's Governance model

Ethereum, the successor to Bitcoin, has been extremely successful at delivering not only the core premise of Bitcoin (a pseudonymous international payment network) but also a robust foundation for future financial infrastructure . Additionally, it uses an effective bicameral system of governance. First, a non-profit foundation led largely by technical architects proposes, with a miner driven consensus mechanism that ratifies. This allows Ethereum to employ dedicated

staff and implement periodic technological upgrades. It also allows occasional interventionist elements.

4 How the Network Stack Solves Hard Problems in Macroeconomics

4.1 Asset-backed currency as a solution to the fiat problem

Since Bretton Woods II and the abandonment of the gold standard the global economy has not had a stable international unit of account. Among other problems, central banks can effectively print money at will, which leads to a certain instability in international relations and the possibility of enormous trade deficits financed by debt and money printing. First proposed by John Law a few centuries ago, the idea of a currency backed by the hardest and most stable of hard assets was previously only theoretically possible, with most historically stable currencies backed instead by precious metals. However, decentralized organizations existing on blockchains can purchase land in multiple legal jurisdictions, tokenize the land, and use it as a stable unit of account.

4.2 Trust networks as a solution to lowest common denominator markets

Markets often fall victim to the 'Keynesian Beauty Problem' and effectively serve as predictive mechanisms for the most commonly held position rather than the optimal one.[7] This may be effectively solved via a multi-track reputational system. This idea was first circulated by noted Venture Capitalist Brock Pierce at Burning Man as a "Trust Currency." A canonical global reputational currency so described would create strong incentives for socially positive behavior.

4.3 Smart contracts as an automated international legal system

Smart contracts as articulated by Nick Szabo and implemented on an Ethereum-type blockchain allow programmatic settlement of embedded logic. While not foolproof, they may be combined with nested chains of intention, much like existed in the legal system of the middle ages. In particular, smart contracts can be augmented with an international arbitration network which resolves cases not easily governed via fully automated fashion.

4.4 Direct democratic engagement as a solution to olicharchic system capture

Populism is often a response to oligarchic capture which has at various points created totalitarian regimes which further enhance oligarchic capture by sup-

pression of contrary point of view. Global transparent ledger systems which support governance creates optimal conditions for capturing public opinion and channeling it into action.

4.5 Network fee as a solution to taxes

Virtually all legacy governance systems depend on the forcible extraction of wealth (e.g. taxes). A system which dynamically and programmatically passes a fee upwards to the governance layer to handle things at the network level allows a transition away from the need for any coercive mechanisms for wealth transfer.

4.6 Basic income as a solution to inefficient or nonexistent welfare

Various regions like Norway and Alaska pass through proceeds of revenues to their citizens. A network state can do the same thing to its members, distributing some amount of proceeds realized through its services back to its constituents.

4.7 An open economy as a solution to wealth concentration

Network governance allows open contributions and open rewards without any preference for large existing stakeholders or other biases based on location or genetics. Additionally, as measured by the Gini index Ethereum appears to have improved substantially over Bitcoin and the US economy. [8]

4.8 Long-term research for near-sighted markets

Investment long-term research including various technologies that may at some point present commercial activities is a classic activity of the nation-state and, yet, one that seems to be waning over the past decade with various high value technological innovations (e.g. blockchain) coming almost entirely from the freelance and private sector as opposed to facilitated by traditional organizations like DARPA. The trust currency provides an additional layer of accountability and rewards people who have a consistent and positive track record of contributions.

5 The Network State

5.1 Implications of the network stack for politics

The network stack allows a new type of political organization that is not dependent on national borders and which can fulfill some of the same functions in

an opt-in manner without any of the coercive value capture mechanisms used in less transparent systems. It is transparent, open, and extensible by design.

```
(network
    (gov network-democracy)
    (legal decentralized-arbitration)
    (bank (asset-backing land)))
```

5.2 Components of the Network State

network (network)

Citizenship and a sense of belonging and duty are two classic assumptions of classic states but the two do not always go hand in hand. In particular, humans have developed overlapping senses of belonging and duty associated with digital communities which augment and often supersede the degree of allegiance at the local level. Additionally the networks of the future must accommodate non-human actors. As a virtual entity, the network state accommodates multiple forms of network allegiance and embedded values.

decision making system (gov network-democracy)

The first principle for our new network state is be the governance mechanism by which it operates. In this case we provide the same option we have suggested elsewhere, while mentioning that this is but one of many possible options. For example, one could provide a clear one citizen one vote liquid democracy system, a constitutional monarchy, a futarchy (e.g. merkle DAO), or some other hybrid system. [9]

legal system (legal decentralized-arbitration)

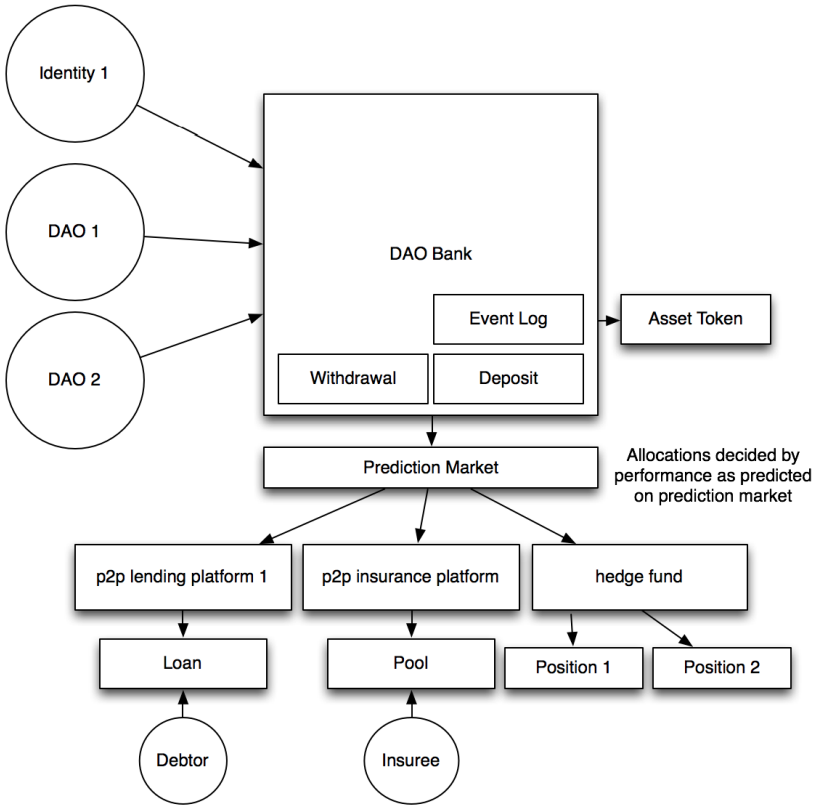
The presence of arbitration in the event either of non-functional code or disputed delivery of real goods and services is a necessary component of an overall legal system. While here is suggested a global arbitration network which, ideally, is integrated into the existing body of law, it is equally possible to propose an entirely novel system or to simply inherit an existing nation state system as a failsafe. This potentially follows the holonic principle of nesting with possibly various tiers of decision making.

banking (bank (asset-backing land))

As with other software architectures, our banking system by default inherits the governing system of its immediate peer which makes it community owned by default. The ostensible values of currency are that it serves as unit of account, store of value, and means of exchange.

More complex examples are possible:

```
(bank
    (deposit)
    (withdrawal)
    (event-log)
    (lending (prediction-market (p2p-platform-1 p2p-platform-2))
```



In this case, a bank engages in lending and programatically distributes funds to other blockchain based organizations.

5.3 Politics and permanence

> Our new Constitution is now established, and has an appearance that promises permanency; but in this world nothing can be said to be certain, except death and taxes.
>
> *Benjamin Franklin*

> We want to overcome the state of nature. It is true that you can say that death is natural, but it is also natural to fight death.
>
> *Peter Thiel*

Where constraints exist it is natural for humans to want to fight all things that limit us, including both death and the seeming inevitability of systems of forced wealth distribution. It seems that we have the possibility of, if not ending such a system out right, at least creating a new system with many of the same benefits without its detriments in a way that is most conducive to overall well-being. The fundamental structure of governance has not been disrupted for over 200 years. It is about time.

6 A Trust Revolution?

> God forbid we should ever be twenty years without such a rebellion[10]
>
> *Thomas Jefferson*

A single revolution implies a state change that may or may not lead to a better status quo, but a perpetual revolution, such as envisioned by Thomas Jefferson, implies a network that is in a constant evolving state of learning, growth, and change. We believe with this model we have a model for a governmental and financial system that is systematically more robust and properly aligned than the current one. Additionally, it creates a global network of trust without much of the abuse potential found in the current one.

7 Acknowledgments

Although the current formulation of trust networks is my own, Brock Pierce (and the Big Imagination camp at Burning Man '16), Matan Field (Backfeed), Adam Apollo and Harlan Wood (trust.exchange), were all instrumental in catalyzing conceptions of trust networks. Initial feedback from Steve Waldman, Nathan Schneider [reviewer names here]

References

[1] Kevin Kelly. Out Of Control: The New Biology Of Machines, Social Systems, And The Economic World, 1994.

[2] Harlan Wood and Joel Dietz. Trust exchange documentation. `http://trust.exchange`.

[3] Arthur Koestler. The sleepwalkers: A history of man's changing vision of the universe, 1959.

[4] Ken Wilber. *A Theory of Everything: An Integral Vision for Business, Politics, Science and Spirituality*. 2001.

[5] Joel Dietz and Marion Vogel. Love Nest Guide. `https://docs.google.com/document/u/1/d/19rraBCqvhrpDgZbRWK97Jxu4D-BCTYhCB83P-6g1KbA`.

[6] Nick Szabo. Bit Gold. `http://unenumerated.blogspot.com/2005/12/bit-gold.html`, 2008.

[7] Robert Shiller. On wall st., a keynesian beauty contest. *New York Times*, 2011.

[8] Vitalik Buterin. Ether Sale: A Statistical Overview. `https://blog.ethereum.org/2014/08/08/ether-sale-a-statistical-overview/`, 2014.

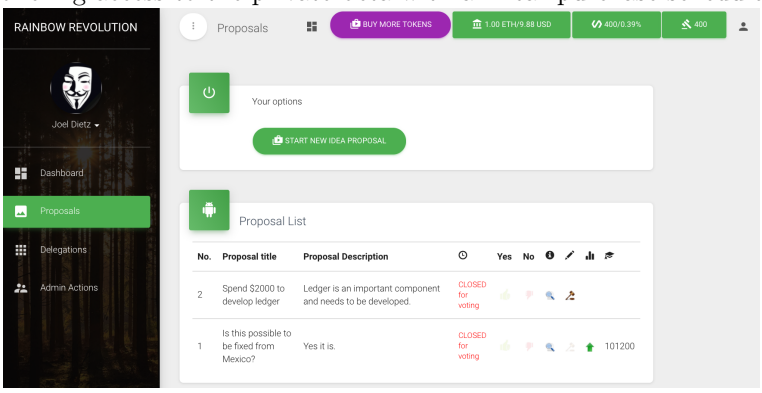
[9] Ralph Merkle. DAO Democracy (annotated by Joel Dietz). `http://fractastical.github.io/dao_democracy.html`, 2016.

[10] Thomas Jefferson. Letter to william stephens smith, 1787.

A Appendix: Implementation

A.1 Current active interface

The Network State has a fully operational model that has been undergoing alpha testing since November '16. With the release of this whitepaper we will be offering access to the private beta with a linear purchase schedule.



This illustrates the principles of network democracy, including purchasing of stake, stake-based voting, and voting delegation.

Live demo at http://liquid.qhoss.com

A.2 Token Contract

```
pragma solidity \^0.4.6;

/// @dev limited token contract
contract MyToken {
    /* Public variables of the token */
    string public standard = 'Token 0.1';
    string public name;
    string public symbol;
    uint8 public decimals;
    uint256 public totalSupply;
    address public owner;
    address public ldAddress;

    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;


    /* This generates a public event on the blockchain that will
        notify clients */
    event Transfer(address indexed from, address indexed to,
       uint256 value);

    /* Initializes contract with initial supply tokens to the
       creator of the contract */
    function MyToken(
        uint256 initialSupply,
        string tokenName,
        uint8 decimalUnits,
        string tokenSymbol,
        address liqiuidDemocracyAddress
        ) {
        balanceOf[msg.sender] = initialSupply;              //
            Give the creator all initial tokens
        totalSupply = initialSupply;                        //
            Update total supply
        name = tokenName;                                   //
            Set the name for display purposes
        symbol = tokenSymbol;                               //
            Set the symbol for display purposes
        decimals = decimalUnits;
        ldAddress = liqiuidDemocracyAddress;
                                    // Amount of decimals for
                                        display purposes
```

```
        if (!msg.sender.send(msg.value))
        throw;                              // Send back any ether sent
            accidentally

         owner = msg.sender;
    }

    /// @dev Used to create new tokens. It can be only called by
        owner or the Liquid Democracy
    /// contract
    /// @param target - address of the member receiving tokens
    /// @param  mintedAmount - amount of new tokens created.
    function mintToken(address target, uint256 mintedAmount)  {

        if (owner != msg.sender ||  ldAddress != msg.sender)
            throw;
        balanceOf[target] += mintedAmount;
        totalSupply += mintedAmount;
        Transfer(0, target, mintedAmount);

    }

     ///@dev destroy the contract when not needed
     function kill() {
         if (msg.sender == owner) selfdestruct(owner);
}

    /* This unnamed function is called whenever someone tries to
        send ether to it */
    function () {
        throw;     // Prevents accidental sending of ether
    }
}
```

A.3 Network Democracy Contract

```
 pragma solidity ^0.4.6;

//Contract token is found in its own file.
//These here are interfaces to access functions and variable of
   token contract.
contract token {

            // balance of tokens for individual member
            mapping (address => uint256) public balanceOf;
```

```
            // this function creates new tokens and assigns it
               to the purchaser.
            // It can be only called by the owner of itself or
               from functions
            // in this contract which is registered with MyToken
                contract.

            function mintToken (address target, uint256
               mintedAmount);
}

// @notice a contract which is inherited by
// main Association contract. owned holds several housekeeping
   functions
contract owned {
    address public owner;


    /// @notice constructor, sets the owner of the contract
    function owned() {
        owner = msg.sender;
    }

    /// @notice modifier to be used in functions, which can be
       only called
    /// by the owner, otherwise call to function will be thrown.
    modifier onlyOwner {
        if (msg.sender != owner) throw;
        _;
    }

    /// @notice used to transfer Ownership
    /// @param newOwner - new owner of the contract
    function transferOwnership(address newOwner) onlyOwner {
        owner = newOwner;

    }

    /// @dev this function will allow on self destruction of
       this contract.
    function kill() {
        if (msg.sender == owner) selfdestruct(owner);
    }
}
```

```
/// @dev Liquid Democracy contract. Allows new members to be
    registered and
/// aquire tokens. Number of acquired tokens also represents
    user voting power.
/// Tokens are held in standard token contract defined here.
contract ld is owned {

    // How long debate should be held in minutes
    uint public debatingPeriodInMinutes;
    // proposals array
    Proposal[] public proposals;
    // to keep number of proposals for easy access
    uint public numProposals;
    // to keep numbrer of members for easy access
    uint public numMembers;
    // to retrieve a member position in the array without
        searching for it
    mapping (address => uint) public memberId;
    // to keep vote weight of each member
    mapping (address => uint256) public voteWeight;
    // keeps delegated votes for each member
    DelegatedVote[] public delegatedVotes;
    // list of registered members
    Member[] public members;
    // address of token contract
    token public sharesTokenAddress;
    // total number of tokens in circulation
    uint public tokensInCirculation;
    // cost of token in wei
    uint public singleTokenCost;
    // date of tokens sale start date in Uinx timestamp
    uint public tokenSaleStartDate;
    // date of tokens sale end date in Unix timestamp
    uint public tokenSaleEndDate;
    // minimum quorum
    uint public minimumQuorum;



    // to store member info
    struct Member {
        // the address of member
        address member;
        // true if member can vote
```

```
        bool canVote;
        // date member created
        uint memberSince;
        // first name of member
        string firstName;
        // last name of member
        string lastName;
        // email address of member
        string userID;
        // true if user has delegated their vote
        bool delegated;
        // for verification when logging in
        // email address hashed with password    f
        bytes32 memberHash;
        // true if user is admin
        bool admin;
        // if provided referral address is stored
        address referral;
    }

    // to store votes delegated by user to another user
    struct DelegatedVote {
        // address of the nominee
        address nominee;
        // address of the voter
        address voter;
        // amount of vote credits, each token is 1 credit
        uint weight;
    }

    // to store proposal info
    struct Proposal {
        // address of the proposal beneficiary
        address recipient;
        // amount of the proposal for beneficiary
        uint amount;
        // description of the proposal
        string description;
        // title of the proposal
        string title;
        // creator of the proposal
        address creator;
        // Unix timestamp of voting deadline
        uint votingDeadline;
        // true if proposal has passed voting deadline and
            tallying has been run
```

```
        uint executed;
        // number of votes cast for this proposal, votes are
            number of total voting weights of participating
            members
        uint numberOfVotes;
        // used for security to  check if user knows the
            combination of info used in hash
        bytes32 proposalHash;
        // array of votes for and against the proposal
        Vote[] votes;
        // list of voting status of members
        mapping (address => bool) voted;
        // implemented to store string of proposal stats.
        // this was due to the fact that this structure reached
            limit of elements allowed by solidity
        // to still accomplish this task, number of variables
            are stored in the string and then
        // parsed when needed
        string proposalStats;

    }

    //  to keep info about voting of each member for the
        proposal
    struct Vote {
        // true if user voted for proposal
        bool inSupport;
        // address of the voter
        address voter;
    }




    // triggered when new proposal is added
    event ProposalAdded(uint proposalID, address recipient, uint
        amount, string description, string title);
    // triggered when vote is cast
    event Voted(uint proposalID, bool position, address voter);
    // triggered when votes on proposals are tallied
    event ProposalTallied(uint proposalID, uint yea , uint nay,
        uint quorum, uint executed);
    // triggered when rules for voting are changed
    event ChangeOfRules(uint minimumQuorum, uint
        debatingPeriodInMinutes);
```

```
// triggered when new member is created or updated
event MembershipChanged(address member, bool isMember,
    string firstName, string lastName, string userID, address
     memberReferral);
// triggered when votes are deleted by a mebmber
event Delegated(address nominatedAddress, address voters,
    uint voteIndex);
// triggered when delegations are reset by admim
event DelegationReset(bool status);
// triggered when tokens are purchased
event BuyTokens(uint numOfTokens, address buyer, uint value)
    ;
// executed when meber account "canVote" is changed
event BlockUnblockMember(address member, bool status);
// executed when ownership of this contract is transferred
event OwnershipTransfer(bool result);
// executed when member cancels vote delegation
event CancelDelegation(address  nominatedAddress, address
    voter, uint voteWeight);
// executed when user buys new tokens or reduces them  and
    their vote weight is updated
event VoteWeightUpdated(address member, uint weightAdded,
    uint totalWeight);
// executed when tokens price is changed
event TokenParmsChange(uint startDate, uint endDate, uint
    tokenPrice);

/* modifier that allows only shareholders to participate in
    auction */
modifier onlyShareholders() {
    if (sharesTokenAddress.balanceOf(msg.sender) == 0) throw
        ;
        _;
}



/// @dev This is constructor function. It allows to
    initialize token address
/// token cost, token sale start date, token sale end date
/// @param minimumSharesToPassAVote - min quorum
/// @param minutesForDebate - number of minutes after which
    debate should expire
/// @param sharesAddress - address of token contract
/// @param tokenCost - cost of token in wei
```

```
function Association(uint minimumSharesToPassAVote, uint
    minutesForDebate,  token sharesAddress, uint tokenCost) {
    changeVotingRules(minimumSharesToPassAVote,
        minutesForDebate);
    sharesTokenAddress = sharesAddress;
    singleTokenCost =  tokenCost;
    tokenSaleStartDate = now;
    tokenSaleEndDate = now + 30 days;

}

/// @dev this function allows on changing minimum quorum and
     length of debate on proposal
/// @param minimumSharesToPassAVote - minimum quorum
/// @param minutesForDebate - length of debate in minutes

function changeVotingRules( uint minimumSharesToPassAVote,
    uint minutesForDebate) onlyOwner {

    if (minimumSharesToPassAVote == 0 )
        minimumSharesToPassAVote = 1;
    minimumQuorum = minimumSharesToPassAVote;
    debatingPeriodInMinutes = minutesForDebate;

    ChangeOfRules(minimumQuorum, debatingPeriodInMinutes);
}


/// @dev start new proposal
/// @param beneficiary - an address of the member who will
    receive the benefits of this proposal if it is approved
/// @param etherAmount - amount of ether beneficiary will
    receive if proposal is approved
/// @param proposalDescription - description of the proposal
/// @param proposalTitle - proposal title
/// @param transactionBytecode - can be sent to increase
    security

function newProposal(
    address beneficiary,
    uint etherAmount,
    string proposalDescription,
    string proposalTitle,
    bytes transactionBytecode
)
    onlyShareholders()
```

```
        returns (uint proposalID)
    {

        proposalID = proposals.length++;
        Proposal p = proposals[proposalID];
        p.recipient = beneficiary;
        p.amount = etherAmount;
        p.description = proposalDescription;
        p.title = proposalTitle;
        p.proposalHash = sha3(beneficiary, etherAmount,
            transactionBytecode);
        p.votingDeadline = now + debatingPeriodInMinutes * 1
            minutes;
        p.executed = 0;
        p.numberOfVotes = 0;
        p.creator = msg.sender;


        numProposals = proposalID+1;

    ProposalAdded(proposalID, beneficiary, etherAmount,
        proposalDescription, proposalTitle);
    }



    /// @dev to check if proposal code is matching the params.
    /// @param proposalNumber - position of proposal in array,
        starts with 0
    /// @param beneficiary -  beneficiary of the proposal
    /// @param etherAmount - ether amount for the beneficiary
    /// @param transactionBytecode
    /// @return  - True if code checks out
    function checkProposalCode(
        uint proposalNumber,
        address beneficiary,
        uint etherAmount,
        bytes transactionBytecode
    )
        constant
        returns (bool codeChecksOut)
    {
        Proposal p = proposals[proposalNumber];
        return p.proposalHash == sha3(beneficiary, etherAmount,
            transactionBytecode);
    }
```

```
/// @dev it allows to change price of token and sale start
    and end dates
/// @param start - Unix time stamp of the token sale start
    date
/// @param end - Unix time stamp of the token sale end date
/// @param tokenPrice - token price in wei

function changeTokenParms(uint start, uint end, uint
    tokenPrice){


    if (start != 0)   tokenSaleStartDate = start;
    if (end !=0)  tokenSaleEndDate = end;
    if (tokenPrice !=0)  singleTokenCost =tokenPrice ;
    TokenParmsChange(start, end, tokenPrice);

}

/// @dev allows on transferring ownership of this contract
/// @param newOwner - address of the new owner

function transferOwnership(address newOwner) onlyOwner {

    // update member records
    members[memberId[newOwner]].admin = true;
    members[memberId[msg.sender]].admin = false;

    // call base contract
    owned.transferOwnership(newOwner);
    OwnershipTransfer(true);
}



/// @dev it saves the vote of the member
/// @param proposalNumber - position of the proposal in
    array
/// @param supportsProposal - true if in favor of proposal
/// @return uint - new vote id

function vote(uint proposalNumber, bool supportsProposal)


    onlyShareholders()
```

```
        returns (uint voteID)
    {
        Proposal p = proposals[proposalNumber];
        if (p.voted[msg.sender] == true || p.executed > 0)
            throw;

        voteID = p.votes.length++;
        p.votes[voteID] = Vote({inSupport: supportsProposal,
            voter: msg.sender});
        p.voted[msg.sender] = true;
        p.numberOfVotes = voteID +1;
        Voted(proposalNumber, supportsProposal, msg.sender);
        return voteID;
    }


    /// @dev helper function returning status of user vote for
        proposal
    /// @param proposalNumber - position of proposal in array
    /// @param voter - address of voter
    /// @return bool - value of voting status, true if voted,
        false if didn't vote
    function hasVoted(uint proposalNumber, address voter)
        constant returns (bool){

        Proposal p = proposals[proposalNumber];
        return p.voted[voter] ;
    }

    /// @dev helper function returning number of votes for
        proposal
    /// @param proposalNumber - position of proposal in array
    /// @return uint - number of votes for proposal
    function numOfVotes(uint proposalNumber) constant returns (
        uint){

        Proposal p = proposals[proposalNumber];
        return p.votes.length;
    }

    /// @dev returns choice of the voter for proposal
    /// @param proposalNumber - position of proposal in the
        array
    /// @param voter - address of a member to retrieve voting
        status for
```

```
/// @return bool - true if member voted for the proposal and
     and false if against it

function howVoted(uint proposalNumber, address voter)
    constant returns (bool){

    Proposal p = proposals[proposalNumber];

    for (uint i = 0; i < p.votes.length; ++i) {
        Vote v = p.votes[i];

        if (v.voter == voter) return v.inSupport;
    }
}

/// @dev helper function to return status if delegated
    member voted already
/// @param proposalNumber - position of proposal in array
/// @param voter - address of a member to retrieve status of
     vote
/// @return bool - true if delegate has voted and false if
    didn't
function hasDelegateVoted(uint proposalNumber, address voter
    ) constant returns (bool){

    for (uint i = 0; i < delegatedVotes.length; i++){
        if (delegatedVotes[i].voter== voter){
            uint id = memberId[delegatedVotes[i].nominee];
            Member m = members[id];
            if (m.delegated){
                hasDelegateVoted(proposalNumber,
                    delegatedVotes[i].nominee);
            }
            else {
                return  hasVoted(proposalNumber,
                    delegatedVotes[i].nominee );
            }
        }
    }
}



/// @dev returns choice of the delegate vote  for proposal
/// @param  proposalNumber - position of proposal in the
    array
```

```
/// @param voter - address of a delegate to retrieve voting
    status for
/// @return bool - true if delegate voted for the proposal
    and and false if against it

function howDelegateVoted(uint proposalNumber, address voter
    ) constant returns (bool){
     for (uint i = 0; i < delegatedVotes.length; i++){
          if (delegatedVotes[i].voter== voter){
               uint id = memberId[delegatedVotes[i].nominee];
               Member m = members[id];
               if (m.delegated){
                    howDelegateVoted(proposalNumber,
                        delegatedVotes[i].nominee);
               }
               else{
                        return howVoted(proposalNumber,
                            delegatedVotes[i].nominee );
               }
          }
     }

}


/// @dev used to check current voting status of proposal
    before the votes are tallied
/// @param proposalNumber - position of proposal in array
/// @return string - concatenated list of parameters with
    their values separated by semicolons ":" and commas

function calculateVotes(uint proposalNumber) constant
    returns (string){


uint quorum = 0;
uint votes = 0;
uint yea = 0;
uint nay = 0;
uint totalMemberCount = members.length;

Proposal p = proposals[proposalNumber];

for (uint i = 0; i <  p.votes.length; ++i) {
     Vote v = p.votes[i];
     uint voteWeightTmp = voteWeight[v.voter];
```

```
        votes += voteWeightTmp ;
        if (v.inSupport) {
            yea += voteWeightTmp ;
            }
            else {
                nay += voteWeightTmp ;
            }
        }

    quorum = votes * 100/ tokensInCirculation;

    string memory tempString = strConcat( "{'yea':",
        uintToString(yea), ", 'nay':", uintToString(nay));
            tempString = strConcat( tempString, ", 'quorum':",
                uintToString(quorum), ", 'votes':");
            tempString = strConcat( tempString, uintToString(
                votes), "}", "");

    return tempString;
    }


    /// @dev tallies proposal votes and saves it. It sets status
         of proposal to executed.
    /// @param proposalNumber - position of proposal in array
    /// @param transactionBytecode - to increase security
    /// @return result - concatenated string of values of voting
         status
    function executeProposal(uint proposalNumber, bytes
        transactionBytecode) returns (uint256 result) {
        Proposal p = proposals[proposalNumber];
        /* Check if the proposal can be executed */
        if (now < p.votingDeadline  /* has the voting deadline
            arrived? */
            || p.executed  > 0      /* has it been already
                executed? */
            || p.proposalHash != sha3(p.recipient, p.amount,
                transactionBytecode)) /* Does the transaction
                code match the proposal? */
            throw;

        /* tally the votes */
    uint quorum = 0;
    uint votes = 0;
    uint yea = 0;
```

```
uint nay = 0;
uint totalMemberCount = members.length;


    for (uint i = 0; i < p.votes.length; ++i) {
        Vote v = p.votes[i];
        uint voteWeightTmp = voteWeight[v.voter];

        votes += voteWeightTmp ;
        if (v.inSupport) {
            yea += voteWeightTmp ;
        } else {
            nay += voteWeightTmp ;
        }
    }
    quorum = votes * 100/ tokensInCirculation;
    /* execute result */
    if (quorum >= minimumQuorum) {

    if (yea > nay ) {
        /* has quorum and was approved */
        p.executed = 1;
        }
    else {
        // it was executed but didn't pass
        p.executed = 2;
        }
    }
    string memory tempString = strConcat( "{'yea':",
        uintToString(yea), ", 'nay':", uintToString(nay));
    tempString = strConcat( tempString, ", 'quorum':",
        uintToString(quorum), ", 'votes':");
    tempString = strConcat( tempString, uintToString(votes),
        "}", "");
    p.proposalStats = tempString;

// Fire Events
ProposalTallied(proposalNumber, yea, nay, quorum, p.executed
    );
result = p.executed;
}


/// @dev facilitates buying of tokens
/// @param numOfTokens - number of tokens to purchased
/// @return bool - true if executed
```

```
function buyTokens(uint numOfTokens) payable returns (bool){

    if (now < tokenSaleStartDate || now > tokenSaleEndDate )
        throw;

    if (msg.sender.balance == 0) throw;

    uint totalTokenCost = singleTokenCost * numOfTokens;
    uint userBalance = msg.sender.balance ;
    uint maxTokenToBuy = userBalance / singleTokenCost;

    if ( numOfTokens >= maxTokenToBuy || totalTokenCost >
       msg.value){
            BuyTokens(0, msg.sender, msg.value);
            throw;
            }

    sharesTokenAddress.mintToken(msg.sender, numOfTokens);
    tokensInCirculation += numOfTokens;

    if (!updateVoteWeight( msg.sender, numOfTokens)) throw;
    BuyTokens(numOfTokens, msg.sender, msg.value);

    return true;
}


/// @dev updates vote weight based on number of purchased
    tokens
/// @param member - address of member to update the vote
    weight
/// @param numTokens - number of tokens purchased indicating
     vote weight
/// @return bool - true if executed

function updateVoteWeight(address member, uint numTokens)
    private returns (bool success){

            voteWeight[member] += numTokens;
            VoteWeightUpdated(member, numTokens, voteWeight[
                member]);
            return true;
}
```

```
    /// @dev allows to cancel the delegation. It will cancel
        delegation on several
    /// levels through use of recursive call.
    /// @param voter - address of the member who wants to cancel
         delegation
    /// @param index - position in array of delegated votes
    /// @param first - true if this is first iteration and index
         needs to be determined

    function removeDelegation(address voter, uint index, bool
        first) {

        uint id;

        for (uint i = 0; i < delegatedVotes.length; i++){

            if (delegatedVotes[i].voter== voter){
                uint idNominee =  memberId[delegatedVotes[i].
                    nominee];
                Member n = members[idNominee];
                if (n.delegated){
                    removeDelegation(delegatedVotes[i].nominee,
                        index, false);
                }
                else{
                    if (first) index = i;

                    DelegatedVote nv = delegatedVotes[i];
                    DelegatedVote vv = delegatedVotes[index];
                    voteWeight[nv.nominee] -= vv.weight ;
                    voteWeight[vv.voter] += vv.weight ;
                    id = memberId[vv.voter];
                    Member m = members[id];
                    m.delegated = false;
                    CancelDelegation(nv.nominee, voter, vv.
                        weight);
                    delete delegatedVotes[index];
                }
            }
        }
    }


    /// @dev allows to delgate votes to another member
    /// @param nominatedAddress - address of the member to
        delegate vote to
```

```
        /// @return voteIndex - position in the array of the
            delegatio record
        function delegate(address nominatedAddress) returns (uint
            voteIndex) {

            uint id;

            uint weight = 0;
            id = memberId[msg.sender];
            Member m = members[id];
            //don't allow members delegation to themselves
            if (nominatedAddress != msg.sender){
                //test if member is not banned
                if (m.canVote){
                    //check if member hasn't delegated their vote
                        yet
                    if (!m.delegated){

                        weight = voteWeight[msg.sender] ;
                        voteWeight[msg.sender] -= weight;
                        voteWeight[nominatedAddress] += weight;
                        m.delegated = true;
                        //mark delegating member as not delegated in
                             case he/she delegated their votes before
                             himself
                        id = memberId[nominatedAddress];
                        Member n = members[id];
                        n.delegated = false;

                        //check if this first delegation and handle
                            resizing of array appropriately
                        if (delegatedVotes.length == 1 &&
                            delegatedVotes[0].nominee == 0   ){
                            delegatedVotes[delegatedVotes.length -1]
                                 = DelegatedVote({nominee:
                                nominatedAddress, voter: msg.sender,
                                weight:weight});
                        }
                        else {
                            delegatedVotes.length ++;
                            delegatedVotes[delegatedVotes.length -1]
                                 = DelegatedVote({nominee:
                                nominatedAddress, voter: msg.sender,
                                weight:weight});
                        }
                    }
```

```
            }
        }
        voteIndex = delegatedVotes.length -1;
        Delegated( nominatedAddress, msg.sender , voteIndex);
    }



    /// @dev admin can reset all delegation to 0
    /// return bool - true if executed

    function resetDelegation() onlyOwner returns (bool result)
    {
        for (uint i=0; i< members.length; i++) {
                voteWeight[members[i].member] =
                    sharesTokenAddress.balanceOf(members[i].
                    member);
                members[i].delegated= false;
            }
        delete delegatedVotes;
        DelegationReset(true);
        return true;
    }


    /// @dev admin can set flag of members ability to vote to
        false or true
    /// @param targetMember - address of a member to set the
        value of "canVote"
    /// @param  canVote - true if member is allowed to vote,
        false otherwise
    function blockUnblockMember(address targetMember, bool
        canVote) onlyOwner {

        uint id;
        id = memberId[targetMember];
        Member m = members[id];
        m.canVote = canVote;
        BlockUnblockMember(targetMember, canVote);
    }

    /// @dev to create new member. Function checks if member
        with this email address exists and if
    /// it doesn't it creats new member.
    /// @param targetMember - address of the new member
    /// @param canVote - sets this flag initial value
```

```
/// @param firstName -
/// @param lastName -
/// @param userID - email address
/// @param memberHash - email address and password hash to
    login
/// @param tokenNum - number of free tokens to assign if any
/// @param memberReferral - referral of the member

function newMember(address targetMember, bool canVote,
    string firstName, string lastName, string userID,
    bytes32 memberHash, uint tokenNum, address memberReferral
    ) {


    uint id;
    bool delegated = false;
    bool adminFlag = false;



    if (stringsEqualMemory("admin@admin.com", userID)){
        adminFlag = true;}


    if(getMemberByUserID(userID) >= 0){
        throw;


    }

    else if (voteWeight[targetMember]==0) {

        memberId[targetMember] = members.length ;
        id = members.length++;
        members[id] = Member({member: targetMember, canVote:
            canVote, memberSince: now, firstName: firstName,
            lastName:lastName, userID:userID, delegated:
            false, memberHash:memberHash, admin:adminFlag,
            referral:memberReferral});
        voteWeight[targetMember]=0;
        numMembers++;

        sharesTokenAddress.mintToken(targetMember, tokenNum)
            ;
        tokensInCirculation += tokenNum;
        updateVoteWeight( targetMember, tokenNum);
```

```
        }
        MembershipChanged(targetMember, canVote, firstName,
            lastName, userID, memberReferral);

    }

    /// @dev used to login user into their account. To check if
        given user exists
    /// @param userID - user email address
    /// @return int - member position in the array
    function getMemberByUserID(string userID) constant returns (
        int memberPosition){

        if (members.length == 0) {
            return -1;
        }

        for (uint i=0; i < members.length; i++){
            if (stringsEqual(members[i].userID , userID) ){
                return int(i);
            }

        }
    return -1;

    }


    /// @notice to compare string when one is in memory and
        other in storage
    /// @param _a Storage string
    /// @param _b Memory string

    function stringsEqual(string storage _a, string memory _b)
        constant internal returns (bool) {
                bytes storage a = bytes(_a);
                bytes memory b = bytes(_b);
                if (a.length != b.length)
                        return false;
                // @todo unroll this loop
                for (uint i = 0; i < a.length; i ++)
                        if (a[i] != b[i])
                                return false;
                return true;
        }
```

```
/// @notice  to compare strings which both reside in memory
/// @param _a Memory string
/// @param _b Memory string
function stringsEqualMemory(string memory _a, string memory
    _b) internal returns (bool) {

    bytes memory a = bytes(_a);
    bytes memory b = bytes(_b);
    if (a.length != b.length)
        return false;
        // @todo unroll this loop
    for (uint i = 0; i < a.length; i ++)
        if (a[i] != b[i])
            return false;
        return true;
}


/// @dev helper function to concatenate strings

function strConcat(string _a, string _b, string _c, string
    _d, string _e) internal constant returns (string){
    bytes memory _ba = bytes(_a);
    bytes memory _bb = bytes(_b);
    bytes memory _bc = bytes(_c);
    bytes memory _bd = bytes(_d);
    bytes memory _be = bytes(_e);
    string memory abcde = new string(_ba.length + _bb.length
        + _bc.length + _bd.length + _be.length);
    bytes memory babcde = bytes(abcde);
    uint k = 0;
    for (uint i = 0; i < _ba.length; i++) babcde[k++] = _ba[
        i];
    for (i = 0; i < _bb.length; i++) babcde[k++] = _bb[i];
    for (i = 0; i < _bc.length; i++) babcde[k++] = _bc[i];
    for (i = 0; i < _bd.length; i++) babcde[k++] = _bd[i];
    for (i = 0; i < _be.length; i++) babcde[k++] = _be[i];
    return string(babcde);
}

function strConcat(string _a, string _b, string _c, string
    _d) internal constant returns (string) {
    return strConcat(_a, _b, _c, _d, "");
}
```

```
function strConcat(string _a, string _b, string _c) internal
    returns (string) {
    return strConcat(_a, _b, _c, "", "");
}

function strConcat(string _a, string _b) internal constant
   returns (string) {
    return strConcat(_a, _b, "", "", "");
}



/// @dev convert uint to string

function uintToString(uint a) internal constant returns (
   string){

    bytes32 st = uintToBytes(a);
    return bytes32ToString(st);
}

/// @dev convert uint to Bytes
function uintToBytes(uint v) internal constant returns (
   bytes32 ret) {
    if (v == 0) {
        ret = '0';
    }
    else {
        while (v > 0) {
            ret = bytes32(uint(ret) / (2 ** 8));
            ret |= bytes32(((v % 10) + 48) * 2 ** (8 * 31));
            v /= 10;
        }
    }
    return ret;
}

/// @dev convert bytes32 to String
function bytes32ToString(bytes32 x) internal constant
   returns (string) {
    bytes memory bytesString = new bytes(32);
    uint charCount = 0;
    for (uint j = 0; j < 32; j++) {
        byte char = byte(bytes32(uint(x) * 2 ** (8 * j)));
        if (char != 0) {
            bytesString[charCount] = char;
```

```
                charCount++;
            }
        }
        bytes memory bytesStringTrimmed = new bytes(charCount);
        for (j = 0; j < charCount; j++) {
            bytesStringTrimmed[j] = bytesString[j];
        }
        return string(bytesStringTrimmed);
    }


}
```

A.4 Future development plan

From a technical standpoint, funds will be used to develop a canonical version of the trust network, the fully automated self-funding structure, additional features for extensibility and automation. This private beta period is expected to last six to nine months and any changes will be decided by network members . Additionally we will be pursuing solid integrations into the legal system for the possession of real assets. Funds will be governed by the private beta with an non-profit used as a failsafe in case of any non-optimally operating smart contracts.

B Appendix: Similarities and Differences to "the DAO"

The Network State highly resembles "the DAO" in that it enables a fully decentralized investment system.

It has, however, the following key differences from "the DAO" :

1. For the foreseeable future the Network Stack exists within a specific legal container that holds smart contracts accountable to their stated intent.
2. The Network State is designed not simply for decentralization but also for automation and integration with futarchy and other artificial intelligence.
3. The Network State integrates with real assets and has over $100mm in active currently accessible deal opportunities via our partnerships.
4. The Network State has mechanisms for ensuring both short and long-term stake in the project, including decision making that is highly vested in the founding team.
5. The Network State is being deployed along with a prototype reputational system (i.e. trust currency).

Consequently, it is not incorrect to say that the Network Stack represents an upgraded version of the DAO released with a stable governance structure, curated deal opportunities, and training wheels in case something breaks.

C Appendix: Version Control

0.2.0 Initial private release 0.2.1 Add appendixes and move source code to appendix 0.2.2 More formatting 0.2.3 The DAO Appendix 0.2.4 Updated diagrams 0.2.5 Numerous small corrections