OpenCollab:
A Blockchain Based Protocol to Incentivize Open Source Software Development and Governance

Yondon Fu

April 30, 2017

1 Introduction

In today's software dependent society, open source software is everywhere. Parties ranging from large technology corporations like Google to individual hobbyist developers use open source software as the building blocks of their own projects. Tools that a developer once had to build from scratch are now widely available for anyone to use for free on websites like Github. Not only can anyone easily use these software packages, but anyone can also freely access, inspect and alter the source code, tailoring it for his or her own specialized needs.

The implications of democratized access to quality software is wide ranging. The open source web framework Ruby on Rails not only powers popular applications such as Twitter and Github that millions of people rely on everyday, but it also made web application development accessible to a broader audience by abstracting away the details of composing together components such as HTTP request handling, database querying and templating. Given the importance of open source software, the task at hand for technologists is to figure out how to make open source software projects sustainable such that organizations and individuals in the future can continue to rely on them in the future.

The sustainability of an open source software project is tied to project health and support. Project health is determined by how actively and adequately project developers communicate with users such that the project addresses the needs of the community. Project support is determined by the availability of financial and technical resources to develop a project[12]. A sustainable open source software projected needs to be both healthy and supported.

A healthy and supported project optimizes the use of developer time and attention, the scarcest resources in open source software projects. Communication between developers and users in a healthy project informs developers of community needs and issues to potentially focus their time and attention on. Availability of financial and technical resources in a supported project ensures developers are free to allocate their all of their time and attention on project issues. Consequently, in a healthy and supported project, developers can properly allocate their time and attention according to community needs.

If project developers fail to properly allocate their time and attention, users might leave a project in search of alternatives that better suit their needs. Canonical, the company behind the Linux operating system Ubuntu, created fragmentation in the Linux community when it shipped a new version of Ubuntu with the Unity interface rather than the standard GNOME interface[6]. Canonical's failure to properly poll for user opinion and allocate developer time and attention accordingly ultimately hurt the Ubuntu project.

1.1 Bounty Systems

A proposed solution to open source software sustainability is a bounty system for project issues such as the one operated by the website Bountysource[4]. In these systems, users can attach monetary bounties to project issues that are rewarded to contributors that successfully resolve issues. While these bounty systems may help projects attract more contributors, they do not take into account the incentives of maintainers. The work done by maintainers to review and merge in code is just as crucial as the work done by contributors. Consequently, bounty systems only partially help with project support.

Furthermore, bounty systems do not necessarily help with project health. Although in some cases, multiple users placing bounties on an issue might signal the importance the community places on that particular issue, it is also possible for malicious actors to place large bounties on issues that would negatively impact project quality. Such a possibility can place a burden on developers of filtering signal from noise and also introduces the possibility of collusion - a contributor might share a large bounty if a maintainer agrees to merge it into the codebase even if the contributed code is of poor quality. As a result, bounty systems can actualy hamper communication between developers and users leading to unmet community needs. Adding any form of financial compensation to open source software projects needs to take into align the incentives of all parties involved or else perverse incentives might arise leading to malicious behavior that harms the quality of the project.

Lastly, bounty systems operated by websites like Bountysource rely on a centralized entity to facilitate transactions. This reliance on a centralized entity not only results in a central point of failure, but can actually be more costly for users. For example, although users can freely transact within the bounty system, Bountysource charges a 10% withdrawal fee if a user wants to cash out. As a result, users choose between giving up a portion of their monetary rewards and giving up the numerous opportunies to use their monetary rewards for their own benefit outside the bounty system. This withdrawal free discourages users from leaving the system which benefits Bountysource, but harms users.

The advent of cryptocurrencies and blockchains introduce the possibility of embedding the rules for a bounty system and other economic incentives directly into a protocol. Rather than relying on a centralized service to enforce the rules of a system, users can instead use a decentralized service powered by a blockchain protocol. The rules of the protocol would facilitate transactions within a distributed network of users.

1.2 Contributions

The primary contributions of this thesis are the following:

- A command line tool that enables a decentralized Git workflow for developing open source software without relying on a a centralized service like Github (Section 3).
- A proof-of-concept blockchain based protocol to incentivize open source software development and governance (Section 4).

The code for the command line tool is available open source at `https://github.com/yondonfu/opencollab`.

The code for the set of smart contracts implementing the OpenCollab protocol is available open source at `https://github.com/yondonfu/opencollab-contracts`.

The motivation behind these contributions is to push the discussion on how to improve open source software sustainability. In particular, these contributions are attempts to answer the following questions relating to open source software sustainability.

- How can developers poll for user opinion on issue priorization for a project?
- How can a project attract regular contributors?
- How can maintainers be incentivized to carefully review and merge pull requests such that the quality of a project is upheld?
- The set of maintainers for a project should be able to change over time. However, maintainers might be reluctant to leave a project in fear of creating a leadership vacuum. How can incentives be structured such that people want to become maintainers?

A detailed description of these contributions can be found in Sections 3 and 4.

2 Background

Although blockchains have only recently captured the attention of academics and business people, the technical foundations of blockchains actually have a longer history. Blockchains use ideas rom past work by computer science researchers in the areas of distributed consensus, electronic money and digital time-stamping.

2.1 Distributed Consensus

The reliability of a distributed system depends on system proccesses to reach consensus on particular values. We can analyze a distributed system as a replicated state machine consisting of a state machine replicated across multiple processes using a deterministic state transition function to map a set of inputs and the current state to a new state[17]. State transitions are atomic such that they either occur completely or do not occur at all, consistent such that they must be valid mappings of inputs and the current state to a new state and durable such that once they occur, state is permanently updated. Thus, we can consider state transition function inputs as transactions[13].

A system using multiple proccesses is vulnerable to proccess faults that cause unexpected process behavior. Faults can be broadly categorized as fail-stop faults, where processes crash and other proccesses can detect the failure, and Byzantine faults, where processes can exhibit arbitrary and potentially malicious behavior[17]. The problem of handling Byzantine faults is more complex than that of handling fail-stop faults because Byzantine faulty processes can transmit conflicting information to other processes that might not be immediately detected[15].

Consequently, distributed systems require specific consensus algorithms to handle Byzantine faults. One example of a Byzantine fault tolerant consensus algorithm is Practical Byzantine Fault Tolerance (PBFT) which offers system reliability as long as out of n processes there are $\frac{n-1}{3}$ or less faulty processes at any given point in time[10]. However, PBFT cannot guarantee system reliability if $\frac{n-1}{3}$ or more processes collude.

2.2 Electronic Money

The advent of the Internet and the mainstream adoption of computing devices encouraged the development of many forms of electronic money by recording balances electronically on devices.

In 1983, David Chaum introduced a cryptographic protocol for anonymous payments using blind signatures. In Chaum's protocol, a bank uses its private key to sign blinded tokens and payees accept signed tokens by clearing a signed token with the bank[11]. The authenticity of tokens can be guaranteed by verifying the bank's signature on tokens with the bank's public key. The bank also does not know the identity of a payer when clearing a signed token sent by a payee because the token was blinded to obfuscate the amount and sender the bank originally signed the token. Chaum applied this protocol in his DigiCash project.

One of the flaws of DigiCash is that it can only offer durable transactions in exchange for decreased anonymity. Users must present tokens to the bank for verification or else they are vulnerable to double spending attacks since electronic messages can easily be duplicated[9]. Furthermore, reliance on the bank creates a bottleneck for system throughput and a central point of failure. If a bank's private key is compromised, an attacker can use the bank's private key to create counterfeit tokens.

Easily duplicated electronic messages leave electronic money systems vulnerable to denial of service attacks. As a solution, Adam Back proposed using hashcash, a easily verifiable, but difficult to compute cost function to mint tokens[1]. The cost function or *proof-of-work* is based on finding partial hash collisions on the k-bit string 0^k, for which the fastest known algorithm is brute force meaning users must perform a certain amount of work in terms of computing cycles to mint tokens. A proof-of-work requirement discourages electronic message duplication by making message creation costly.

Nick Szabo highlighted the utility of proof-of-work for electronic money in his bit gold protocol. The protocol uses a proof of work function to compute a string of bits that is timestamped in a distributed property title registry[18]. Users can verify owernship of a string of bits in the title registry. In contrast with DigiCash, bit gold allows valuable bits to be created, transferred and stored without depending on a trusted third party. However, a system implementing such a protocol was never implemented in practice.

2.3 Digital Time-Stamping

Widespread digitization of all types of documents brought many benefits to society, but alos introduced the question of how to prove the existence and time of creation or change of a digital document.

In 1991, Haber and Stornetta presented a time-stamping method for digital documents that consisted of certificates cryptographically signed by a time-stamping service. The certificates contain the hash of the document as well as linking information from a previous certificate which includes a hash of the previous certificate's linking information[14]. The result is a hash linked chain of certificates that prevents the faking of time-stamps.

Bayer, Haber and Stornetta extended this time-stamping method using merkle trees. In the original time-stamping method, verification of a document timestamp can require at most N steps by following the chain links to a time-stamp certificate that is trustworthy[2]. Instead of linking N hashes of documents, the hash values can be stored in a merkle tree. Participants can record the hashes of their own documents and the sibling hash values along the path from the document hash to the root of the merkle tree. Consequently, verification can be done in at most $\lg N$ steps by presenting the document hash and the $\lg N$ hashes on the path to the root. This modified time-stamping approach reduces storage requirements and verification time.

2.4 Blockchains

Blockchains combine learnings from past work in distributed consensus, electronic money and digital time-stamping. A blockchain is a type of distributed system with two key defining characteristics. The first characteristic is that transactions are grouped into blocks. A common optimization is to store the transactions in a merkle tree and only include the merkle root in the block reminiscient of Bayer, Haber and Stronetta's digital time-stamping method using merkle trees. This optimization allows for easy verification of the existence of a transaction while decreasing the storage requirements for a block. The second characteristic is that blocks are linked by cryptographic hashes. As demonstrated by Haber and Stornetta's work with hash linked digital timestamps, a hash linked chain of blocks prevents tampering of blocks unless an adversary has majority control of the system such that it can rewrite the entire hash linked chain. The result is a distributed ledger that is not controlled or managed by a central entity powered by a network of connected computers that use a consensus mechanism to reach agreement over shared data[20].

The first blockchain was the Bitcoin blockchain[16]. Bitcoin uses a proof-of-work consensus algorithm based on Adam Back's hashcash cost function. Nodes solve partial hash collisions to propose blocks of transactions. Additionally, Bitcoin also introduced an economic mechanism as a complement to cryptographic primitives to secure distributed systems by rewarding nodes that propose blocks with a economically valuble token - bitcoin.

2.5 Ethereum

Vitalik Buterin developed Ethereum as a solution to leverage the distributed consensus capabilities of blockchains to create decentralized applications. Ethereum is a blockchain with built-in Turing complete programming language that allow users to write so called *smart contracts* that define arbitrary state transition functions[8]. Nodes in the Ethereum network run the Ethereum Virtual Machine (EVM). The value proposition of smart contracts is the ability to define arbitrary rules and agreements in a self-enforcing and self-executing program. As a result, participants in a protocol or network can trust the automatic enforced execution of code in the smart contract rather than trust some centralized entity.

Although security focused members of the computer science community have expressed a fair amount of concern about the viability of Ethereum as a blockchain used for smart contracts due to the large attack surface presented by its Turing complete programming language, the stark reality is that the Ethereum developer ecosystem is the most active of any other blockchain ecosystem. Furthermore, various members of the community are actively researching methods to better secure the Ethereum network including formal verification, proof-of-stake as an alternative consensus algorithm to proof-of-work and smart contract programming languages with stronger security guarantees. Consequently, with these points in mind we decided to build OpenCollab on Ethereum.

3 Decentralized Git Workflow

3.1 Mango

Althought Git is a distributed version control system, it is commonly used in a centralized manner. Developers often work using a local Git repository and coordinate with other developers by pushing their changes to a remote Git repository hosted somewhere. While developers can choose to use their own servers to host remote Git repositories, it is far more common to rely on a web service such as Github to handle hosting. Outsourcing hosting work to Github relieves developers of additional work, but also forces developers to trust and rely on Github.

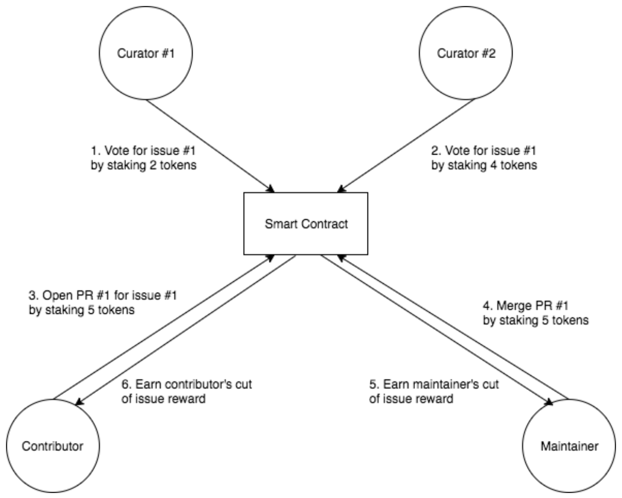
Mango is a remote protocol for Git that uses Ethereum smart contracts for repository access control and stores Git objects in decentralized content addressable storage networks[3]. Moving repository control away from a centralized third party to a smart contract also introduces the possibility of new repository features such as programatic payments and voting mechansims. A smart contract controlled repository makes the economic incentives for development and governance in the OpenCollab protocol possible.

3.2 Extensions to Mango

In order for Mango to be viable for a decentralized Git workflow, it needs to support pull requests and issues. As a part of our contributions, we extended the MangoRepo smart contract implementation by adding an issue tracking system and pull request support.

Users create and edit issues by uploading issue contents to Swarm and mapping the Swarm hash to an issue id in the contract.

Users can create a pull request by forking the project and initializing a new Mango repository for the fork. The user can then make relevant changes in the fork, push the fork using Mango and reference the MangoRepo contract address for the fork in a new pull request. Maintainers can then review a pull request by cloning the Mango repository using its contract address.


Curator #1
Curator #2
1. Vote for issue #1 by staking 2 tokens
2. Vote for issue #1 by staking 4 tokens
Smart Contract
3. Open PR #1 for issue #1 by staking 5 tokens
4. Merge PR #1 by staking 5 tokens
6. Earn contributor's cut of issue reward
5. Earn maintainer's cut of issue reward
Contributor
Maintainer

4 OpenCollab Protocol

The OpenCollab protocol is implemented by a set of Ethereum smart contracts. By building on the Ethereum platform, we can rely on security properties of the underlying Ethereum blockchain. With the details of consensus and security abstracted away, the OpenCollab protocol focuses on defining economic incentives and rules that encourage participants maintain and further the quality of an open source software project.

4.1 Protocol Roles

- **Curators**: curate project issues by staking tokens.
- **Contributors**: open pull requests to resolve issues by staking tokens.
- **Maintainers**: review and merge pull requests for issues by staking tokens.

4.2 OpenCollab Token

The OpenCollab token (OCT) powers the OpenCollab protocol. The value offered by the token is influence over an open source software project. Furthermore, the token serves the following purposes in the protocol:

- Used in a staking mechanism for issue curation. Curators stake tokens to signal the importance they place on an issue.
- Used in a staking mechanism for opening pull requests. Contributors stake a certain number of tokens when opening a pull request. If a contributor's pull request is closed without being merged in to the project, the contributor's staked tokens are destroyed. The possibility of losing staked tokens discourages contributors from opening pull requests unless they are confident about the quality of their contributions.
- Used in a staking mechanism for merging pull requests. Maintainers stake a certain number of tokens when they initiate a merge. Before a merge is finalized, a token holder can challenge a maintainer's merge to start a voting round. If token holders decide to veto a maintainer's merge, the maintainer's staked tokens are destroyed. The possibility of losing staked tokens discrouages maintainers from merging pull requests that do not benefit a project. The challenge and voting process for a merge is described in more detail in Section 4.5.
- Used in deposits for token holders that choose to participate as voters in protocol governance.

An initial allocation of tokens will be distributed so that the various protocol roles can be fulfilled by token holders. A project creator can initialize a Mango repository and mint a certain amount of tokens for the initial allocation. The initial allocation might be done using a token crowdsale or by disbursement at the discretion of the project creator.

OCT is an ERC20 compliant token[21] and is divisible by 10^{18}. When a repository using the OpenCollab protocol is created, a smart contract is created governing a OCT that is specific to that particular repository. As a result, there can be many repository specific versions of OCT.

4.3 Curating Issues

Curators stake a number of tokens to an issue to signal the importance that they place on the issue. Since curators lock up their tokens for a period of time when they stake tokens to an issue, they have limited curation power. Curators exchange either funds for tokens by purchasing them on the secondary market or work for tokens by providing work to the project as a contributor or maintainer. Furthermore, since curators take on the risk of a fall in token value, they have skin in the game[19]. If curators signal importance for bad issues, developers poorly allocate their time and attention. If developers properly allocate their time and attention on resolving issues that would increase the quality of a project, the community might lose interest and less people would desire influence over the project leading to a fall in token value. Consequently, curators have an incentive to signal importance for issues that accurately reflect the needs of the community. As well curated issues are resolved, the value of the token would increase thereby benefiting curators.

4.4 Opening Pull Requests

Contributors open pull requests by staking contributorStake tokens, where contributorStake is a repository parameter.

If a contributor's pull request is successfully merged by a maintainer, the contributor receives a portion of the issue's token reward. The portion can be calculated as reward - (reward $*$ maintainerPercentage).

If a contributor's pull request is closed without being merged into the project, the contributor's contributorStake staked tokens are destroyed.

4.5 Merging Pull Requests

Maintainers merge pull requests by staking maintainerStake tokens, where maintainerStake is a repository parameter.

If a maintainer wants to merge a pull request, it calls initMergePullRequest(id) to signal an intent to merge a particular pull request and starts a challenge period. During this period, any token holder can challenge the maintainer by calling challenge() and staking challengerStake tokens.

If a maintainer is not challenged during the challenge period, he can call mergePullRequest(id). The maintainer receives a portion of the issues's token reward which can be calculated as reward $*$ maintainerPercentage.

If a maintainer is challenged during the challenge period, a voting period begins. Voting takes place using a two step commit and reveal protocol first formalized by Brassard, Chaum and Crepeau[5]. During the commit step, voters with a minimum voterDeposit deposit in the smart contract vote to uphold or veto a maintainer's merge by calling commitVote(hash) with the cryptographic hash of their vote and a secret phrase. A vote to uphold is a 0 and a vote to veto is a 1. The secret phrase can be any random string only known to the voter. The value of the vote is secure from an attacker as long as only the voter knows the secret phrase used when generating the hash. We use the keccak256 hashing function since it is used by Ethereum.

During the reveal step, voters reveal the values of their votes by submitting the concatenation of their vote and secret phrase used in the commit step by calling revealVote(vote). The smart contract verifies that the submitted vote corresponds with the committed hash and tallies up votes as voters reveal them. Finally, anyone can call voteResult() which compares the number of uphold and veto votes. The value that receives the majority of vote ($\geq 50\%$) wins. Voters on the losing side of the vote are penalized such that voterPenaltyPercentage is deducted from their deposits. Voters on the winning side of the vote are rewarded such that voterRewardPercentage is added to their deposits.

If a maintainer's merge decision is upheld, the maintainer is able to call mergePullRequest(id) to finalize the merge. The challenger's challengerStake staked tokens are destroyed and the maintainer can claim his portion of the issue token reward.

If a maintainer's merge decision is vetoed, the challenger's staked tokens are returned and the maintainer's maintainerStake tokens are destroyed and is removed from the maintainer set for the repository. Consequently, the former maintainer would not only lose the staked tokens, but also the economic value of future issue token rewards. The possibility of losing tokens and maintainer status serves to encourage maintainer to only merge pull requests that ensure the quality of the project.

During a voting period, a malicious maintainer might attempt to bribe voteres to vote to uphold the merge decision. However, given N voters, a maintainer would have to bribe $N/2$ voters to ensure the merge decision is upheld. Since a two step commit and reveal protocol is used for voting, voters cannot know with certainty that other voters will accept the maintainer's bribe. Furthermore, since a voter will be penalized if he is on the losing side of the vote and rewarded if he is on the winning side of the vote, the bribe would need to be greater than both the possible reward and penalty. An economically rational maintainer would not try to bribe voters to finalize a merge as long as the cost of the bribe multiplied across $N/2$ voters is greater than the issue token reward a maintainer stands to gain from merging in the pull request.

5 Conclusion

References

[1] Adam Back. *Hashcash - A Denial of Service Counter-Measure.* http://www.hashcash.org/papers/hashcash.pdf. Accessed: 2017-4-29. 2002.

[2] Dave Bayer, Stuart Haber., and W. Scott Stornetta. “Improving the Efficiency and Reliability of Digital Time-Stamping”. In: *Sequences II: Methods in Communication, Security and Computer Science.* 1993, pp. 329–334.

[3] Alex Beregszaszi. *Mango: Git, completely decentralized.* https://github.com/axic/mango/. Accessed: 2017-04-24.

[4] *Bounty Source: Support for Open-Source Software.* https://www.bountysource.com/. Accessed: 2017-04-24.

[5] Gilles Brassard, David Chaum, and Claude Crepeau. “Minimum Disclosure Proofs of Knowledge”. In: *Journal of Computer and System Sciences* 37 (1988), pp. 156–189.

[6] Jon Brodkin. *Ubuntu Unity is dead: Desktop will switch back to GNOME next year.* https://arstechnica.co.uk/information-technology/2017/04/ubuntu-unity-is-dead-back-to-gnome/. 2017.

[7] Ethan Buchman. “Tendermint: Byzantine Fault Tolerance in the Age of Blockchains”. MA thesis. The University of Guelph, 2016.

[8] Vitalik Buterin. *Ethereum White Paper: A Next-Generation Smart Contract and Decentralized Application Platform.* https://github.com/ethereum/wiki/wiki/White-Paper. 2013.

[9] L. Jean Camp, Marvin Sirbu, and J.D. Tygar. “Token and Notational Money in Electronic Commerce”. In: *Proceedings of the 1st USENIX Workshop on Electronic Commerce.* 1995, pp. 1–12.

[10] Miguel Castro and Barbara Liskov. “Practical Byzantine Fault Tolerance and Proactive Recovery”. In: *ACM Transactions on Computer Systems* 20.4 (2002), pp. 398–461.

[11] David Chaum. “Blind Signatures for Untraceable Payments”. In: *Advances in Cryptology: Proceedings of Crypto 82.* Springer US, 1983, pp. 199–203.

[12] Nadia Eghbal. *What success really looks like in open source.* https://medium.com/@nayafia/what-success-really-looks-like-in-open-source-2dd1facaf91c. 2016.

[13] Jim Gray. “The Transaction Concept: Virtues and Limitations (Invited Paper)”. In: *Proceedings of the Seventh International Conference on Very Large Data Bases.* Vol. 7. 1981, pp. 144–154.

[14] Stuart Haber and W. Scott Stornetta. “How to time-stamp a digital document”. In: *Journal of Cryptology* 3.2 (1991), pp. 9–111.

[15] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine Generals Problem”. In: *ACM Transactions on Programming Languages and Systems* 4.3 (1982), pp. 382–401.

[16] Satoshi. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System.* https://bitcoin.org/bitcoin.pdf. 2008.

[17] Fred B. Schneider. “Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial”. In: *ACM Computing Surveys (CSUR)* 22.11 (1990), pp. 299–319.

[18] Nick Szabo. *Bit gold.* http://unenumerated.blogspot.com/2005/12/bit-gold.html. Accessed: 2017-4-29. 2008.

[19] Nassim N. Taleb and Constantine Sandis. “The Skin In The Game Heuristic for Protection Against Tail Events”. In: *Review of Behavioral Economics* 1 (2014), pp. 1–21.

[20] Peter Van Valkenburgh.
What is "Blockchain" anyway?
https://coincenter.org/entry/what-is-blockchain-anyway.
2017.

[21] Fabian Vogelsteller. *ERC: Token Standard.*
https://github.com/ethereum/EIPs/issues/20.
Accessed: 2017-4-28.